PHP5 Orientado a Objetos
(Básico)

Índice

Primeira Parte – Programação Orientada a Objetos com PHP5

1) Introdução

A orientação a objetos é atualmente o paradigma de programação mais aceito e considerado moderno, pois apareceu juntamente dom os padrões de projeto para resolver problemas da programação procedural.

Atualmente as empresas que trabalham com programação, ao contratar programadores estão exigindo, uma formatura ou estudando na área, o conhecimento de programação orientada a objetos e algumas exigem o conhecimento de um dos bons frameworks.

Conhecer programação orientada a objetos, padrões de projeto e bons frameworks valoriza profissionais de programação.

A programação orientada a objetos tem uma grande preocupação em esconder o que não é importante para o usuário e em realçar o que é importante.

Atualmente o PHP5 oferece um suporte muito rico em termos de OO.
Com o PHP5 o PHP ganhou uma completa infraestrutura de orientação a objetos.

A OO é uma programação toda voltada para código modular.

Vale lembrar que orientação a objetos não é uma linguagem, mas sim uma forma de programar, um paradigma, que é implementado por várias das linguagens modernas atuais.

Programação Orientada a Objetos - é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

Em alguns contextos, prefere-se usar modelagem orientada ao objeto, em vez de programação.
A análise e projeto orientados a objetos têm como meta identificar o melhor conjunto de objetos para descrever um sistema de software. O funcionamento deste sistema se dá através do relacionamento e troca de mensagens entre estes objetos.

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. (Wikipédia).

Aqui foi percorrido o caminho inverso para o aprendizado da Orientação a Objetos em PHP. Primeiro foi um curso de um dos melhores frameworks em PHP (CakePHP), onde eu podia criar com grande facilidade um aplicativo em PHP orientado a objetos com bons padrões, examinar o código e procurar entender. Depois, mesmo conseguindo criar aplicativos em PHPOO, percebi que precisava sentir terra nos pés, sentir mais segurança de como as coisas estavam sendo feitas, ou seja, precisava entender de MVC e também de Programação Orientada a Objetos. Bem, agora chegou a vez do MVC e da base geral chamada Orientação a Objetos com PHP5.
Lembrando que somente falo de PHP5 porque a equipe já trabalha no 6 e não mais atualiza o 4, que deve ser abandonado por todos, sem contar que não tem a maioria dos recursos abordados aqui.

2) Programação Procedural x Orientada a Objetos

PHP é hoje uma das mais populares linguagens de programação web.
Mais de 60% dos servidores web rodam Apache com PHP.

Uma das razões da popularidade do PHP é sua baixa curva de aprendizado.
Qualquer um pode escrever código PHP sem seguir nenhuma convenção e misturar as camadas de apresentação com a de negócios. Isso acontecendo em um grande projeto por vários anos pode se transformar em inimaginável monstro.

A Programação Orientada a Objetos (POO) é uma boa prática de programação para a criação e gerenciamento de projetos mais facilmente.

A POO estimula qualquer linguagem para uma melhor codificação, para melhor performance e para escrever projetos muito grandes sem se preocupar muito sobre o gerenciamento.
POO elimina os aborrecimentos e dificuldades do gerenciamento de grandes aplicações.

Identificando Classes, Objetos, Propriedades e Métodos

Vale lembrar que uma classe contém atributos e os atributos são propriedades e métodos.
Tomemos como exemplo um carro:
Propriedades – número de portas, cor, preço, marca, modelo, ano, etc
Métodos – buzinar, dirigir, virar à esquerda, partir, parar, etc

Lembrar também que classes manipulam definições e objetos manipulam valores.

3) História do PHP

No Início

Em 1995, Rasmus Lerdorf começou a desenvolver o PHP/FI.
Ele não imaginou que sua criação acabaria por levar ao desenvolvimento do PHP que nós conhecemos hoje, que está sendo usado por milhões de pessoas. A primeira versão do "PHP/FI", chamado Personal Homepage Tools/ Form Interpreter, foi uma coleção de scripts Perl em 1995(1). Um dos básica recursos foi algo parecido com Perl, para manipulação de envios de formulários, mas faltou muitas características úteis comun em uma linguagem, como laços.

(1) http://groups.google.com/group/comp.infosystems.www.authoring.cgi/msg/cc7d43454d64d133?pli=1

Onde ele anunciou o PHP Tools (mensagem histórica transcrita abaixo):

Mensagem sobre o tópico Announce: Personal Home Page Tools (PHP Tools)
O grupo no qual você está postando é um grupo da Usenet. As mensagens postadas neste grupo farão com que o seu e-mail fique visível para qualquer pessoa na internet.

Com o objetivo de verificação, digite os caracteres que você vê na figura abaixo ou os números que ouvir ao clicar no ícone de acessibilidade. Ouça e digite os números que ouvir

Rasmus Lerdorf
Mais opções 8 jun 1995, 04:00

These tools are a set of small tight cgi binaries written in C.
They perform a number of functions including:

. Logging accesses to your pages in your own private log files
. Real-time viewing of log information
. Providing a nice interface to this log information
. Displaying last access information right on your pages
. Full daily and total access counters
. Banning access to users based on their domain
. Password protecting pages based on users' domains
. Tracking accesses ** based on users' e-mail addresses **
. Tracking referring URL's - HTTP_REFERER support
. Performing server-side includes without needing server support for it
. Ability to not log accesses from certain domains (ie. your own)
. Easily create and display forms
. Ability to use form information in following documents

Here is what you don't need to use these tools:

. You do not need root access - install in your ~/public_html dir
. You do not need server-side includes enabled in your server
. You do not need access to Perl or Tcl or any other script interpreter

. You do not need access to the httpd log files

The only requirement for these tools to work is that you have
the ability to execute your own cgi programs. Ask your system
administrator if you are not sure what this means.

The tools also allow you to implement a guestbook or any other
form that needs to write information and display it to users
later in about 2 minutes.

The tools are in the public domain distributed under the GNU
Public License. Yes, that means they are free!

For a complete demonstration of these tools, point your browser
at: http://www.io.org/~rasmus

--
Rasmus Lerdorf
ras...@io.org
http://www.io.org/~rasmus

PHP/FI 2

O PHP/FI foi reescrito em 22 em 1997, mas naquele tempo o desenvolvimento foi quase exclusivamente do Rasmus.
Após o seu lançamento em novembro do mesmo ano, Andi Gutmans e Zeev Suraski esbarraram no PHP/FI, enquanto procuravam uma linguagem para desenvolver uma solução em comércio eletrônico como um projeto universitário.
Eles descobriram que o PHP / FI não era tão poderoso quanto parecia, e que faltavam muitas características comuns.
Um dos aspectos mais interessantes incluidos foi a forma como o laço while foi implementado. O analisador léxico varre o script e quando encontra a palavra-chave while, ele se lembra da sua posição no arquivo. No final do laço, o ponteiro do arquivo procurado volta ao posição salva, e o ciclo completo foi relido e re-executado.

PHP 3
Zeev e Andi decidiram reescrever completamente a linguagem de script.
Eles então fizeram parceria com Rasmus para liberar a versão PHP 3, e junto veio também um novo nome: PHP: Hypertext Preprocessor, para enfatizar que o PHP era um produto diferente e não adequado apenas para uso pessoal. Zeev e Andi também tinham concebido e executado uma nova extensão da API. Esta nova API permitiu facilmente adicionar novas extensões para realizar tarefas como acessar bancos de dados, verificadores ortográficos e outras tecnologias, o que atraiu muitos desenvolvedores que não faziam parte do "Core Group" para participar e contribuir com o projeto PHP. No momento do release 3 do PHP, em Junho de 1998, a base instalada estimada em PHP consistia em cerca de 50.000 domínios. PHP 3 deflagrou o início da descoberta real do PHP, e foi o primeira versão a ter uma base instalada de mais de um milhão de domínios.

PHP 4

No final de 1998, Zeev e Andi olharam para traz, para os seus trabalhos em PHP 3 e sentiram que poderia ter escrito a linguagem de script ainda melhor, assim começaram a escrever novamente o PHP Enquanto o PHP 3 faz continuamente o parse dos scripts enquanto executando, PHP 4 veio com um novo paradigma de "primeiro compilar e executar mais tarde". A etapa de compilação não compila scripts PHP em código de máquina, mas ao invés compila-los em byte code, que é então executado pelo Zend Engine (Zend representa Zeev e Andi), o novo coração do PHP 4. Devido a esta nova forma de execução de scripts, o desempenho do PHP 4 foi muito melhor do que do PHP 3, com apenas uma pequena quantidade de compatibilidade com versões anteriores (4). Entre outras melhorias era uma extensão da API melhorada para melhor desempenho de run-time, uma camada de abstração de servidor web que permite ao PHP 4 rodar nos mais populares servidores web, e muito mais. PHP 4 foi lançado oficialmente em 22 de maio de 2002, e hoje (2004) a sua base instalada já ultrapassou 15 milhões domínios.
Em 2007 estava com 23 milhões.

Detalhes em: http://www.php.net/usage.php

2 http://groups.google.com/groups?selm=Dn1JM9.61t%40gpu.utcc.utoronto.ca.
3 http://groups.google.com/groups?selm=Pine.WNT.3.96.980606130654.-317675I-100000%40shell.lerdorf.on.ca.
4 http://www.php.net/manual/en/migration4.php.

No PHP 3, o número de versão menor (o dígito do meio) nunca era usado, e todas as versões foram contadas como 3.0.x. Isso mudou no PHP 4, e os menores número de versão foram utilizados para designar alterações importantes na linguagem. A primeiro importante mudança veio na versão PHP 4.1.0 (5) que introduziu superglobals como $ _GET e $ _POST. Superglobais podem ser acessados de dentro de funções sem ter que usar a palavra global. Esse recurso foi adicionado com o fim de permitir a register_globals opção INI para ser desligada. register_globals é um recurso no PHP que converte automaticamente as variáveis de entrada como "?foo = bar" em http://php.net/?foo=bar para uma variável do PHP chamado $foo. Porque muitas pessoas não checam entradas de variáveis corretamente, muitas aplicações tiveram brechas de segurança, que tornou muito fácil de burlar a segurança e código de autenticação.

Com a nova superglobals em vigor, em 22 de abril de 2002, foi o PHP 4.2.0 lançado com o register_globals desativado por padrão. PHP 4.3.0, a última versão significativa do PHP 4, foi lançada em 27 de dezembro de 2002. Esta versão introduziu a Command Line Interface (CLI), um arquivo modificado e uma camada de I/O de rede (chamada streams), e uma biblioteca GD empacotada. Embora a maioria das adições não tivessem nenhum efeito real sobre os usuários finais, a versão principal foi se "bumped" devido às grandes mudanças no núcleo do PHP.

PHP 5
Logo depois, a demanda por mais recursos comuns na orientação a objetos aumentaram imensamente, e Andi surgiu com a idéia de reescrever a parte de orientação aobjetos da Zend Engine. Zeev e Andi escreveram o documento "Zend Engine II: Característica, Descrições e Design" (6) que iniciou o salto para as discussões acaloradas sobre o futuro do PHP.
Embora a linguagem básica permanecesse a mesmo, muitos características foram adicionados,

outras cairam e outras foram alteradas para um PHP 5 amadurecido. Por exemplo, namespaces e herança múltipla, que foram mencionadas no documento original, nunca apareceram no PHP 5. Herança múltipla foi abandonado em favor de interfaces e namespaces foram abandonadas completamente (até a versão 5.3). Você pode encontrar uma lista completa dos novos recursos no capítulo "O que há de novo no PHP 5?"

PHP 5 é esperado para manter e até aumentar a liderança do PHP no mercado de desenvolvimento web. Não só revoluciona o apoio a orientação a objetos do PHP, mas também contém muitas novas funcionalidades que o tornam a plataforma de desenvolvimento web final. A reescrita das funcionalidades do XML no PHP 5, o coloca em pé de igualdade com outras tecnologias web em algumas áreas e até supera em outras, especialmente devido à nova extensão SimpleXML que torna ridiculamente fácil de manipular documentos XML. Além disso, o novo SOAP, MySQLi, e uma variedade de outras extensões são importantes suportes do PHP para tecnologias adicionais.

5 http://www.php.net/release_4_1_0.php.
6 http://zend.com/engine2/ZendEngine-2.0.pdf.

Referência: Traduzido com o auxílio do Translate do Google - http://www.google.com.br/language_tools?hl=pt-BR
do Prefácio do e-book PHP 5 Power Programming - http://www.phptr.com/perens
http://www.informit.com/content/images/013147149X/downloads/013147149X_book.pdf

História da Orientação a Objetos no PHP

Quando o PHP foi criado ele não implementava a OO em si.
Após o PHP/FI, quando Zeev, Rasmus e Andy reescreveram o core e lançaram o PHP3, foi introduzido uma muito básica orientação a objetos.
Quando o PHP4 foi lançado as características do OO amadureceram com algumas características introduzidas.
Mas a equipe reescreveu o core engine e introduziu um modelo de objetos completamente novo no lançamento do PHP5.

O PHP é uma linguagem que nos permite escrever código em dois sabores: procedural e orientado a objetos.

Quando escrevemos uma grande aplicação no estilo procedural ele deverá ficar quase impossível de gerenciar após algumas versões.

A maioria das grandes aplicações é escrita usando o estilo orientado a objetos.

4) Benefícios da POO

A POO foi criada para tornar a vida dos programadores mais fácil.
Com POO podemos quebrar problemas em pedaços menores que serão comparativamente mais fáceis de entender.
O principal objetivo da POO é: tudo que você desejar fazer, faça com objetos. Objetos são basicamente pequenas e discretas peças de código que podem incorporar dados e comportamento neles. Em uma plicação todos os objetos são conectados entre si. Eles compartilham dados para resolver problemas.

- Reusabilidade - DRY 'Don't Repeat Yourself' - Não repita seu código, mas ao contrário reutilize-o.

- Refactoring - Na hora de refazer fica mais simples, visto que temos pedados menores e mais simples.

- Extensível - Podemos criar novos objetos a partir dos existentes, que herdam todas as suas características e adicionar novas.

- Manutenibilidade - São fáceis de manter devido a que seus objetos/partes são pequenos e inclusive permitem facilidade para juntar.

- Eficiência - O conceiro de orientação a objetos é atualmente associado a uma melhor eficiência devido as suas características.
Vários padrões de projeto são criados para melhorar a eficiência.

5) Classe

É um pedaço de código que contém propriedades e métodos.

É semelhante a um array, que armazena dados chamados de chaves e valores.

Classes são mais que arrays, por que contém métodos. Classes também podem ocultar e exibir informações, o que não é possível para os arrays.
Classes também parecem com estruturas de dados e podem incorporar vários outros objetos em si.
Orientação a objetos no PHP5 tem grandes diferenças da POO no PHP4.

Vejamos um exemplo de classe útil e como usar:

```
<?php
//class.emailer.php
class emailer
{
    private $sender;
    private $recipients;
    private $subject;
    private $body;

    function __construct($sender)
    {
        $this->sender = $sender;
        $this->recipients = array();
    }

    public function addRecipients($recipient)
    {
        array_push($this->recipients, $recipient);
    }

    public function setSubject($subject)
    {
        $this->subject = $subject;
    }

    public function setBody($body)
    {
        $this->body = $body;
    }
    public function sendEmail()
    {
        foreach ($this->recipients as $recipient)
        {
            $result = mail($recipient, $this->subject, $this->body,      "From: {$this->sender}\r\n");
            if ($result) echo "Mail successfully sent to {$recipient}<br/>";
```

```
        }
    }
}
?>

<?php
// Exemplo de uso:
$emailer = new emailer("ribafs@gmail.com"); // Consrutor
$emailer->addRecipients("tiago@ribafs.org"); // Acessando o método e passando dados
$emailer->setSubject("Apenas um teste do Curso PHP5OO");
$emailer->SetBody("Olá Tiago, como vai meu amigo?");
$emailer->sendEmail();
?>
```

6) Diferenças entre a POO no PHP4 e no PHP5

Orientação a objetos em PHP5 tem grande diferença em relação ao PHP4.

A POO em PHP4 é pobre e inteiramente aberta, sem qualquer restrição de uso das propriedades e métodos. Não podemos usar os modificadores public, protected e private para os métodos e propriedades.

Em PHP podemos encontrar interfaces mas não recursos como abstract e final.

Uma **interface** é uma peça de código que qualquer classe pode implementar e significa que a classe precisa ter todos os métodos declarados na interface. Precisamos implementar todos os métodos que existem na interface. Na interface podemos declarar apenas o nome e o tipo de acesso dos métodos. Uma interface só pode extender outra interface.

Uma classe **abstrata** (abstract) é onde alguns métodos podem ter algum corpo também. Então qualquer classe pode extender esta classe abstrata e extender também todos os seus métodos na classe abstrata.

Uma classe **final** é uma classe que não podemos extender.
Em PHP5 podemos usar todas essas.

Em PHP4 não existe herança múltipla por interfaces.
No PHP5 múltiplas heranças são suportadas através da implementação de múltiplas interfaces.

Em PHP4 qualquer coisa é static. Isso significa que se você declara qualquer método na classe, você pode chamar este diretamente sem criar uma instância da classe.
Por exemplo:

```
<?php
class Abc
{
    var $ab;

    function abc()
    {
        $this->ab = 7;
    }

    function mostrealgo()
    {
        echo $this->ab;
    }
}

abc::mostrealgo();
?>
```

Rodando este código no PHP4 ele funciona, mas no PHP5 acusa erro, pois o this não vale numa chamada static.

No PHP4 não existe:

- Constante de classe;
- Propriedade static;
- Destrutor.
- Exceptions

Existe sobrecarga de métodos via métodos mágicos como __get() e __set() no PHP5.

7) Iniciando com a Programação Orientada a Objetos no PHP5

Vamos analisar a classe Emailer:

```
<?php
//class.emailer.php
class emailer
{
    private $sender;
    private $recipients;
    private $subject;
    private $body;

    function __construct($sender)
    {
        $this->sender = $sender;
        $this->recipients = array();
    }

    public function addRecipients($recipient)
    {
        array_push($this->recipients, $recipient);
    }

    public function setSubject($subject)
    {
        $this->subject = $subject;
    }

    public function setBody($body)
    {
        $this->body = $body;
    }

    public function sendEmail()
    {
        foreach ($this->recipients as $recipient)
        {
            $result = mail($recipient, $this->subject, $this->body,     "From: {$this->sender}\r\n");
            if ($result) echo "Mail successfully sent to {$recipient}<br/>";
        }
    }
}
?>
```

Temos aí quatro propriedades, todas com visibilidade tipo private. Isso é uma das recomendações, que as propriedades sejam desse tipo para que ninguém tenha acesso a elas diretamente.
Para ter acesso devemos criar os métodos getter e setter.

Depois temos um método construtor, que recebe um parâmetro e retorna o remetente e os destinatários (recipients). Cada vez que essa classe for instanciada deveremos passar o remetente como parâmetro e receberemos o mesmo e os destinatários.

Observe que o método construtor não tem modificador de visibilidade, portanto assume o padrão, que é public.

Depois temos mais quatro métodos, todos publics:

addRecipients($recipient)
setSubject($subject)
setBody($body)
sendEmail()

Observe seus nomes, camelCase e iniciando com minúsculas.

Se todos os métodos são public, significa que ao instanciar essa classe teremos acesso a todos os métodos, mas não às propriedades da classe, que são private.

Antes de usar uma classe precisamos instanciá-la. Após instanciar podemos acessar suas propriedades e métodos usando o operador -> após o nome da instância.

Veja o exemplo de uso abaixo da nossa classe Emailer:

```
<?php
include_once('class.emailer.php');
// Exemplo de uso:
$emailer = new emailer("ribafs@gmail.com"); // Consrutor
$emailer->addRecipients("tiago@ribafs.org"); // Acessando o método e passando dados
$emailer->setSubject("Apenas um teste do Curso PHP5OO");
$emailer->SetBody("Olá Tiago, como vai meu amigo?");
$emailer->sendEmail();
?>
```

Veja que incluímos a classe que vamos usar na primeira linha.

Primeiro criamos uma instância chamada $emailer da classe emailer() e passamos um e-mail como parâmetro:
$emailer = new emailer("ribafs@gmail.com"); // Consrutor

Nós passamos o e-mail como parâmetro porque o construtor da classe recebe um e-mail como parâmetro.

Caso não passemos parâmetro ou passemos um número diferente de parâmetros para a classe ao instanciar acontecerá um erro fatal.

8) Modificadores de Acesso

Os modificadores foram introduzidos no PHP5. São palavras-chaves que ajudam a definir como será o acesso das propriedades e métodos quando alguém instanciar a classe.

private – este modificador não permite ser chamado fora da classe, mas de dentro da classe pode ser chamado sem problema qualquer método.
public – este é o modificador default, o que significa que quando um método não tiver modificador ele será public. Public significa que pode ser chamado de fora da classe sem problema.
protected – somente pode ser acessado de uma subclasse, ou seja, de uma classe que extendeu esta.

Vamos a um exemplo do modificador protected:

Abrir a classe Emailer e mudar a propriedade $sender para protected:

```
<?php
//class.emailer.php
class Emailer
{
    protected $sender;  // Mudar aqui
    private $recipients;
    private $subject;
    private $body;

    function __construct($sender)
    {
        $this->sender = $sender;
        $this->recipients = array();
    }

    public function addRecipients($recipient)
    {
        array_push($this->recipients, $recipient);
    }

    public function setSubject($subject)
    {
        $this->subject = $subject;
    }

    public function setBody($body)
    {
        $this->body = $body;
    }

    public function sendEmail()
    {
        foreach ($this->recipients as $recipient)
```

```
        {
            $result = mail($recipient, $this->subject, $this->body,"From: {$this->sender}\r\n");
            if ($result) echo "Mail successfully sent to {$recipient}<br/>";
        }
    }
}
?>
```

Agora criar o arquivo class.extendedemailer.php com o código:

```
<?php
class ExtendedEmailer extends emailer
{
    function __construct(){}

    pubic function setSender($sender)
    {
        $this->sender = $sender;
    }
}
?>
```

Agora testar assim:

```
<?php
include_once("class.emailer.php");
include_once("class.extendedemailer.php");

$xemailer = new ExtendexEmailer();
$xemailer->setSender("joaobrito@joao.com");
$xemailer->setSubject("Teste extendido");
$xemailer->setBody("Olá João, espero que esteja tudo bem com você!");
$xemailer->sendEmail();
?>
```

Observe que a propriedade sender não será acessível de fora dessas duas classes.

9) Construtor

O PHP5 tem dois tipos de construtor, o que usa a palavra reservada __construct() e o compatível com a versão 4 do PHP, que tem o mesmo nome da classe.
Quando acontecer de uma classe tiver dois construtores, um com __construct e outro com o mesmo nome da classe, o __construct será usado e o outro ignorado.

Veja o exemplo a seguir e teste instanciando a classe Fatorial:

```
<?php
//class.factorial.php
class Factorial
{
    private $result = 1;
    private $number;

    function __construct($number)
    {
        $this->number = $number;
        for($i=2; $i<=$number; $i++)
        {
            $this->result*=$i;
        }
        echo "__construct() executed. ";
    }

    function factorial($number)
    {
        $this->number = $number;
        for($i=2; $i<=$number; $i++)
        {
            $this->result*=$i;
        }
        echo "factorial() executed. ";
    }
    public function showResult()
    {
        echo "Factorial of {$this->number} is {$this->result}. ";
    }
}
?>
```

Destrutor

É o método que quando executado destrói o objeto, invocado pela palavra-chave __destruct(). Será invocado automaticamente sempre ao final da execução do script.

Exemplo de uso:

```
function __destruct()
{
    print " O objeto foi destruído.";
}
```

10) Constantes de Classe

No PHP5 para criar uma constante de classe usamos a palavra reservada **const.** Atualmente esta deve funcionar como uma variável static, a diferença é que ela é somente leitura.

Veja este exemplo:

```
<?php
class WordCounter
{
    const ASC=1; //you need not use $ sign before Constants
    const DESC=2;
    private $words;

    function __construct($filename)
    {
        $file_content = file_get_contents($filename);
        $this->words = (array_count_values(str_word_count(strtolower ($file_content),1)));
    }

    public function count($order)
    {
        if ($order==self::ASC)
        asort($this->words);
        else if($order==self::DESC)
        arsort($this->words);
        foreach ($this->words as $key=>$val)
        echo $key ." = ". $val."<br/>";
    }
}
?>
```

Exemplo de uso, com o arquivo words.txt:

```
<?php
include_once("class.wordcounter.php");
$wc = new WordCounter("words.txt");
$wc->count(WordCounter::DESC);
?>
```

Observe que estamos acessando uma propriedade da classe WordCounter (DESC) de fora sem passar pela instância mas diretamente com o operador ::.
Execute e veja que é um bom utilitário.

11) Herança – Extendendo uma Classe

Uma característica forte do POO e que é uma das mais utilizadas, onde podemos criar uma classe inteiramente nova partindo de uma existente. A nova classe pode preservar todas as características da classe pai ou pode sobrescrevê-las. A nova classe (subclasse) também pode adicionar novas funcionalidades.
Nós temos uma classe para envio de e-mails e agora precisamos de uma classe que envie e-mails do tipo HTML. Não precisamos criar uma classe inteiramente nova, podemos partir da existente e adicionar apenas uma funcionalidade que dará suporte ao HTML.

```
<?php
class HtmlEmailer extends emailer
{
    public function sendHTMLEmail()
    {
        foreach ($this->recipients as $recipient)
        {
            $headers = 'MIME-Version: 1.0' . "\r\n";
            $headers .= 'Content-type: text/html; charset=iso-8859-1' . "\r\n";
            $headers .= 'From: {$this->sender}' . "\r\n";
            $result = mail($recipient, $this->subject, $this->body, $headers);
            if ($result) echo "HTML Mail successfully sent to {$recipient}<br/>";
        }
    }
}
?>
```

A classe HtmlEmailer é uma subclasse da classe Emailer
e a classe Emailer é uma superclasse da classe HtmlEmailer.

Veja que a classe HtmlEmailer extendeu a classe Emailer e adicionou um método sendHTMLEmail() e pode continuar usando todos os métodos da classe pai.

Alerta: caso a subclasse não tenha construtor então será invocado o construtor da superclasse.

Não podemos extender mais que uma classe de cada vez. Herança múltipla somente é suportada com interfaces.

Exemplo de uso da classe HtmlEmailer:

```
<?php
include_once("class.htmlemailer.php");
$hm = new HtmlEmailer();
// fazer outras cosias ...
$hm->sendEmail();
$hm->sendHTMLEmail();
?>
```

12) Sobrescrevendo Métodos

Quando se extende uma classe a subclasse pode sobrescrever todos os métodos da superclasse, desde que eles tenham modificadores protected ou public. Para isso basta criar um método com o mesmo nome do que desejamos sobrescrever. Por exemplo, se criarmos um método com o nome sendMail na subclasse HTMLMailer ele irá sobrescrever o método da superclasse. Assim também acontecerá com as propriedades. Se criarmos uma propriedade na subclasse com o mesmo nome de uma na superclasse ela ofuscará a da superclasse.

Prevenindo a Sobrescrição

Se acontecer que ao criar um método não queremos que ela seja sobrescrito nunca então devemos adicionar o modificador **final** a ele, como no exemplo:

public final function nomeMetodo()

Assim ao ser extendida esta classe o método não poderá ser sobrescrito.

Prevenindo para não Extender uma Classe

As classes também tem o mesmo recurso para que não sejam extendidas. Para isso também usamos o modificador **final**:

final class NomeClasse

Esta classe não poderá ser extendida.

13) Polimorfismo

É o princípio que permite que classes derivadas de uma mesma superclasse tenham métodos iguais (mesma assinatura e parâmetros) mas com comportamentos diferentes definidos em cada uma das classes . Este processo é chamado de polimorfismo. É o processo de criar vários objetos de uma classe básica específica.

Exemplo:

```
<?php
include("class.emailer.php");
include("class.extendedemailer.php");
include("class.htmlemailer.php");
$emailer = new Emailer("hasin@somewherein.net");
$extendedemailer = new ExtendedEmailer();
$htmlemailer = new HtmlEmailer("hasin@somewherein.net");
if ($extendedemailer instanceof emailer ) echo "Extended Emailer is Derived from Emailer.<br/>";
if ($htmlemailer instanceof emailer ) echo "HTML Emailer is also Derived from Emailer.<br/>";
if ($emailer instanceof htmlEmailer ) echo "Emailer is Derived from HTMLEmailer.<br/>";
if ($htmlemailer instanceof extendedEmailer ) echo "HTML Emailer is Derived from
```

Emailer.
";
?>
Checar se classe é instância de outra:
operador **instanceof**

14) Interface

Interface é uma classe vazia que contém somente as declarações dos métodos (corpo em branco). Qualquer classe que implemente uma interface precisa conter todas as declarações dos métodos. Uma classe usa uma interface passando a palavra reservada "implements", assim como uma classe usa uma classe pai passando a palavra reservada "extends". Lembrando que nas interfaces podemos apenas declarar métodos mas não podemos escrever o corpo dos métodos, que obrigatoriamente precisam permanecer vazios.

Usada para criar regras para a criação de novas classes.

Vamos tentar mostrar a necessidade das interfaces: supondo que trabalhamos numa empresa coordenando uma equipe de três programadores e queremos criar uma classe de driver para os bancos de dados, onde usamos três SGBDs, MySQL, PostgreSQL e SQLite e queremos deixar a cargo de cada programador um SGBD, cada um criando uma classe para o seu.
Nós queremos que os programadores sempre trabalhem, obrigatoriamente, com dois métodos, connect e execute. Aí é onde entra nossa interface, que conterá a assinatura dos dois métodos, connect e execute, mas os corpos dependerão de cada SGBD, no caso a cargo de um dos programadores.

Veja nossa interface:

```
<?php
//interface.dbdriver.php
interface DBDriver
{
    public function connect();
    public function execute($sql);
}
?>
```

Cada programador criará uma classe para seu SGBD que deve implementar essa interface e agora eles usarão os dois métodos da interface e adicionarão corpo aos mesmos.

Vejamos um exemplo do primeiro programador criando uma classe que implementará a interface DBDriver:

```
<?php
//class.postgresqldriver.php
class PostgreSQLDriver implements DBDriver
{
```

```
}
?>
```

Obrigatoriamente devemos definir os dois métodos da interface e atente para o parâmetro do método execute.

15) Classes Abstratas

Semelhante às interfaces, mas agora os métodos podem conter corpo e não se implementa classes abstratas, mas ao contrário se extende.

Praticamente uma classe abstrata existe para ser extendida.

Abstrair (simplificar, considerar isoladamente). Um sistema OO deve ser um sistema separado em módulos, focando nas peças mais importantes e ignorando as menos importantes (na primeira etapa) para a construção de sistemas robustos e reusáveis.

Exemplo simples:

```
<?php
//abstract.reportgenerator.php
abstract class ReportGenerator
{
    public function generateReport($resultArray)
    {
        // Código do gerator de relatórios
    }
}
```

Porque colocamos um método abstrato nesta classe? Porque a geração de relatórios sempre envolve bancos de dados.

```
<?php
include_once("interface.dbdriver.php");
include_once("abstract.reportgenerator.php");

class MySQLDriver extends ReportGenerator implements DBDriver
{
    public function connect()
    {
        // conectar ao SGBD
    }

    public function execute($query)
    {
        // Executar consulta e mostrar resultado
    }
```

```
    // Não precisamos declarar ou escrever novamente o método reportGenerator aqui, pois foi
    // extendido da classe abstrata
}
?>
```

Observe que podemos implementar uma interface e ao mesmo tempo extender uma classe, como no exemplo acima.

Alerta
Não podemos declarar uma classe abstrata como final, pois a final não pode ser extendida e a abstrata foi criada para ser extendida.

Quando um método foi declarado como abstrato isso significa que a subclasse precisa sobrescrever o método. Um método abstrato pode não conter nenhum corpo onde é definido.

Declaração de um método abstrato:

```
abstract public function connectDB();
```

16) Propriedades e Métodos Static

Para acessar qualquer método ou propriedade de uma classe temos que criar uma instância, ou seja usar: $objeto = new Classe(); De outra forma não podemos acessar. Mas existe uma exceção para métodos e propriedades static. Estes podem ser acessados diretamente sem a criação de instância. Um membro static é como um membro global.

Onde utilizamos um método static?
A criação de novos objetos com instância é algo que pode consumir recursos do computador e um método estático evita isso.

Exemplo:

```
<?php
// class.dbmanager.php

class DBManager
{
    public static function getMySQLDriver()
    {
        // Instanciar o novo objeto do Driver do MySQL e retornar
    }

    public static function getPostgreSQLDriver()
    {
        // Instanciar o novo objeto do Driver do PostgreSQL e retornar
    }
```

```
    public static function getSQLiteDriver()
    {
        // Instanciar o novo objeto do Driver do SQLite e retornar
    }
}
?>
```

Podemos acessar qualquer método static usando o operador :: ao invés do ->.
Veja:

```
<?php
// test.dbmanager.php
include_once("class.dbmanager.php");
$dbdriver = DBManager::getMySQL();
// agora processamos oeprações do banco com o objeto $dbdriver
?>
```

Veja que usamos o operador :: e não precisamos criar nenhuma instância.

Métodos static geralmente executam uma tarefa e finalizam a mesma.

Alerta: não podemos usar ***$this*** com métodos static. Como a classe não é instanciada então $this não existe. Ao contrário podemos usar a palavra reservada ***self***.

Um exemplo de como a propriedade static funciona:

```
<?php
//class.statictester.php
class StaticTester
{
    private static $id=0;

    function __construct()
    {
        self::$id +=1;
    }

    public static function checkIdFromStaticMehod()
    {
        echo "Current Id From Static Method is ".self::$id."\n";
    }

    public function checkIdFromNonStaticMethod()
    {
        echo "Current Id From Non Static Method is ".self::$id."\n";
    }
}
```

```
$st1 = new StaticTester();
StaticTester::checkIdFromStaticMehod();
$st2 = new StaticTester();
$st1->checkIdFromNonStaticMethod(); //returns the val of $id as 2
$st1->checkIdFromStaticMehod();
$st2->checkIdFromNonStaticMethod();
$st3 = new StaticTester();
StaticTester::checkIdFromStaticMehod();
?>
```

Sempre que criamos uma nova instância ela afeta todas as instância pois a propriedade é declarada como static.

Membros static tornam a orientação a objetos no PHP como a antiga programação procedural. Use métodos static com cuidado.

17) Métodos Acessores

Métodos acessores são simplesmente métodos que são devotados somente a receber e setar o valor de qualquer das propriedades de classe. É uma boa prática acessar o valor das propriedades indiretamente através dos métodos acessores.
Existem dois tipos de métodos acessores: os getters (retornam valor de uma propriedade) e os setters (setam o vlaor de uma propriedade).

Exemplo:

```
<?php
//class.student.php
class Student
{
    private $properties = array();

    function __get($property)
    {
        return $this->properties[$property];
    }

    function __set($property, $value)
    {
        $this->properties[$property]="AutoSet {$property} as: ".$value;
    }
}
?>
```

Convenções:

setter:
setNome()

getNome()

Estes métodos podem ajudar a filtrar a entrada de dados antes de configurar no trabalho.

Existem métodos mágicos que fazem esse trabalho de forma automática e reduzem o trabalho em 90%.

18) Métodos Mágicos para as Propriedades da Classe

O processo é chamado de sobrecarga de propriedade.

Os dois métodos mágicos são o __get() e o __set().

Exemplo:

```
<?php
//class.student.php
class Student
{
    private $properties = array();

    function __get($property)
    {
        return $this->properties($property);
    }

    function __set($property, $value)
    {
        return $this->properties($property)="AutoSet {$property} as: ".$value;
    }
}
?>
```

Agora testando:

```
<?php
$st = new Estudante();
$st->name = "Tiago";
$st->email = "tiago@ribafs.org";
echo $st->name."<br>";
echo $st->email;
?>
```

Quando executamos o código anterior o PHP percebe que não existem propriedades com nome name nem email na classe. Desde que as propriedades não existam então o método __set() será chamado atribuindo valor para a propriedade.

19) Método Mágico para Sobrecarregar Método de Classe

O método mágico __cal() ajuda a sobrecarregar qualquer método chamado no contexto do PHP5. Permite prover ações ou retornar valores quando métodos indefinidos são chamados em um objeto. Têm dois argumentos, o nome do método e um array de argumentos passado para o método indefinido.

Exemplo:

```
<?php
class Overloader
{
    function __call($method, $arguments)
    {
        echo "You called a method named {$method} with the following arguments <br/>";
        print_r($arguments);
        echo "<br/>";
    }
}

$ol = new Overloader();
$ol->access(2,3,4);
$ol->notAnyMethod("boo");
?>
```

20) Funções de Informações sobre Classes

get_class_methods

Retorna um vetor com os nomes dos métodos de uma determinada classe.
array get_class_methods (string nome_classe)
Exemplo:

```
<?php
class Funcionario
{
   function SetSalario()
   {
   }
   function GetSalario()
   {
   }
   function SetNome()
   {
   }
   function GetNome()
   {
   }
}

print_r(get_class_methods('Funcionario'));
?>
```

get_class_vars

Retorna um vetor com os nomes das propriedades e conteúdos de uma determinada classe. Note que são valores estáticos definidos na criação da classe.
array get_class_vars (string nome_classe)

```
<?php
class Funcionario
{
   var $Codigo;
   var $Nome;
   var $Salario = 586;
   var $Departamento = 'Contabilidade';
   function SetSalario()
   {
   }
   function GetSalario()
   {
   }
}
print_r(get_class_vars('Funcionario'));
?>
```

get_object_vars

Retorna um vetor com os nomes e conteúdos das propriedades de um objeto. São valores dinâmicos que se alteram de acordo com o ciclo de vida do objeto.

array get_object_vars (object nome_objeto)

Exemplo:

```
<?php
class Funcionario
{
    var $Codigo;
    var $Nome;
    var $Salario = 586;
    var $Departamento = 'Contabilidade';
    function SetSalario()
    {
    }
 function GetSalario()
 {
 }
}
$jose = new Funcionario;
$jose->Codigo = 44;
$jose->Nome = 'José da Silva';
$jose->Salario += 100;
$jose->Departamento = 'Financeiro';
print_r(get_object_vars($jose));
?>
```

get_class

Retorna o nome da classe a qual um objeto pertence.

string get_class (object nome_objeto)

Exemplo:

```
<?php
class Funcionario
{
    var $Codigo;
    var $Nome;
    function SetSalario()
    {
    }
    function GetSalario()
    {
    }
}
```

```
$jose = new Funcionario;
echo get_class($jose);
?>
```

get_parent_class

Retorna o nome da classe ancestral (classe-pai). Se o parâmetro for um objeto, retorna o nome da classe ancestral da classe à qual o objeto pertence. Se o parâmetro for uma string, retorna o nome da classe ancestral da classe passada como parâmetro.

string get_parent_class (mixed objeto)

Parâmetros	Descrição
objeto	Objeto ou nome de uma classe.

Exemplo:

```
<?php
class Funcionario
{
   var $Codigo;
   var $Nome;
}
class Estagiario extends Funcionario
{
   var $Salario;
   var $Bolsa;
}
$jose = new Estagiario;
echo get_parent_class($jose);
echo "\n";      // quebra de linha
echo get_parent_class('estagiario');
?>
```

is_subclass_of

Indica se um determinado objeto ou classe é derivado de uma determinada classe.

boolean is_subclass_of (mixed objeto, string classe)

Parâmetros	Descrição
objeto	Objeto ou nome de uma classe.
classe	Nome de uma classe ancestral.

Exemplo:

```
<?php
class Funcionario
{
```

```
   var $Codigo;
   var $Nome;
}
class Estagiario extends Funcionario
{
   var $Salario;
   var $Bolsa;
}
$jose = new Estagiario;
if (is_subclass_of($jose, 'Funcionario'))
{
   echo "Classe do objeto Jose é derivada de Funcionario";
}
echo "\n";      // quebra de linha
if (is_subclass_of('Estagiario', 'Funcionario'))
{
   echo "Classe Estagiario é derivada de Funcionario";
}
?>
```

method_exists

Verifica se um determinado objeto possui o método descrito. Podemos verificar a existência de um método antes de executar por engano um método inexistente.

boolean method_exists (object objeto, string método)

Parâmetros	Descrição
objeto	Objeto qualquer.
método	Nome de um método do objeto.

Exemplo:

```
<?php
class Funcionario
{
   var $Codigo;
   var $Nome;
   function GetSalario()
   {
   }
   function SetSalario()
   {
   }
}
$jose = new Funcionario;
if (method_exists($jose, 'SetNome'))
{
   echo 'Objeto Jose possui método SetNome()';
}
if (method_exists($jose, 'SetSalario'))
```

```
{
   echo 'Objeto Jose possui método SetSalario()';
}
?>
```

call_user_func

Executa uma função ou um método de uma classe passado como parâmetro. Para executar uma função, basta passar seu nome como uma string, e, para executar um método de um objeto, basta passar o parâmetro como um array contendo na posição 0 o objeto e na posição 1 o método a ser executado. Para executar métodos estáticos, basta passar o nome da classe em vez do objeto na posição 0 do array.
mixed call_user_func (callback função [, mixed parâmetro [, mixed ...]])

Parâmetros	Descrição
função	Função a ser executada.
parâmetro	Parâmetro(s) da função.

Exemplo:

```
<?php
// exemplo chamada simples
function minhafuncao()
{
   echo "minha função! \n";
}
call_user_func('minhafuncao');
// declaração de classe
class MinhaClasse
{
   function MeuMetodo()
   {
      echo "Meu método! \n";
   }
}
// chamada de método estático
call_user_func(array('MinhaClasse', 'MeuMetodo'));
// chamada de método
$obj = new MinhaClasse();
call_user_func(array($obj, 'MeuMetodo'));
?>
```

class_exists

Checa se uma classe existe.

```
<?php
include_once("class.emailer.php");
echo class_exists("Emailer");
// Retorna true se existir e false se o contrário
```

```
?>
```

A melhor maneira de usar a função class_exists() é sempre antes de instanciar uma classe, checando primeiro se ela existe. Então se existir instanciamos, caso contrário disparamos um erro. Assim fica mais estável.

```
<?php
include_once("class.emailer.php");
if( class_exists("Emailer"))
{
    $emailer = new Emailer("joaobrito@joao.com");
}else{
    die ("Classe Emailer não encontrada!");
}
?>
```

get_declared_classes

Procurar classes carregadas atualmente. Retorna um array com as classes carregadas.

```
<?php
include_once("class.emailer.php");
print_r(get_declared_classes());
?>
```

is_a

Checa o tipo de uma classe.

```
<?php
class ClassePai
{

}

class Filha extends ClassPai
{

}

$cc = new Filha();
if (is_a($cc, "Filha")) echo "Esta classe é um objeto do tipo Filha";
echo "<br>";
if (is_a($cc, "ClassePai")) echo "Esta classe é um objeto do tipo ClassePai";
?>
```

21) Tratamento de Exceções

Um recurso muito importante adicionado no PHP5. Com isso ganhamos um tratamento de erros mais eficiente e com mais recursos.

```
<?php
//class.db.php
error_reporting(E_ALL - E_WARNING);
class db
{
    function connect()
    {
        if (!pg_connect("host=localhost password=pass user=username dbname=db"))
            throw new Exception("Cannot connect to the database");
    }
}

    $db = new db();
    try {
        $db->connect();
    }
    catch (Exception $e)
    {
        print_r($e);
    }
?>
```

Usando um bloco try ... catch podemos capturar todos os error. Podemos usar blocos dentro de blocos. Veja um exemplo:

```
<?php
include_once("PGSQLConnectionException.class.php");
include_once("PGSQLQueryException.class.php");
error_reporting(0);
class DAL
{
    public $connection;
    public $result;

    public function connect($ConnectionString)
    {
        $this->connection = pg_connect($ConnectionString);
        if ($this->connection==false)
        {
            throw new PGSQLConnectionException($this->connection);
        }
    }
    public function execute($query)
    {
```

```
        $this->result = pg_query($this->connection,$query);
        if (!is_resource($this->result))
        {
            throw new PGSQLQueryException($this->connection);
        }
        //else do the necessary works
    }
}

$db = new DAL();
try{
        $db->connect("dbname=golpo user=postgres2");
    try{
        $db->execute("select * from abc");
    }

    catch (Exception $queryexception)
    {
        echo $queryexception->getMessage();
    }
}
catch(Exception $connectionexception)
{
    echo $connectionexception->getMessage();
}
?>
```

Se o código não conseguir conectar ao SGBD ele captura o erro e exibe a mensagem:
Sorry, couldn't connect to PostgreSQL server.

Se a conexão for bem sucedida mas existir problema na consulta, então exibirá uma mensagem adequada.

Veja que para uma falha na consão usamos o objeto PGSQLConnectionException e para a falha na consulta usamos o objeto PGSQLQueryException.

Podemos personalizar o desenvolvimento desses objetos extendendo a classe core Exception do PHP5. Veja os exemplos:

```
<?php
Class PGSQLConnectionException extends Exception
{
    public function __construct()
    {
        $message = "Desculpe, impossível conectar ao servidor do PostgreSQL!";
        parent::__construct($message, 0000);
    }
}
?>
```

Agora a classe PGSQLQueryException:

```
<?php
Class PGSQLQueryException extends Exception
{
    public function __construct($connection)
    {
        parent::__construct(pg_last_error($connection), 0);
    }
}
?>
```

Podemos capturar todos os erros como exceptions, exceto os erros fatais. Veja o código:

```
<?php
function exceptions_error_handler($severity, $message, $filename, $lineno) {
    throw new ErrorException ($message, 0, $severity, $filename, $lineno);
}

set_error_handler('exception_error_handler');
?>
```

Autor: fjoggen@gmail.com no manual do PHP.

22) Convenções para Nomes

- Cada classe deve ficar em um script php e somente uma classe
- Classe de nome Emailer fica no arquivo class.emailer.php.
- Todos os arquivos em minúsculas e nomes de classes em CamelCase. Nomes de classes e arquivos não podem ininiar com algarismos.
- Sugestão para os vários nomes:
  - class.emailer.php
  - interface.emailer.php
  - abstract.emailer.php
  - final.emailer.php

- Nomes de propriedades e métodos em camelCase, mas iniciando com minúsculas: sendEmail().

23) Modelando algumas Classes

Pessoa
Propriedades: nome, cpf, altura, endereco, telefone, uf, cidade
Métodos (funcionalidades): nascer, crescer, trabalhar, sorrir, passear

Carros
Propriedades: cor, ano, placas, marca, modelo
Métodos: buzinar, funcionar, frear, partir

Computadores
Propriedades: processador, memoria, teclado, gabinete, monitor, ano, fabricante, marca, modelo
Métodos: iniciar, funcionar, processar, acessar

Contas
Propriedades: cpf_conta, titular, banco, agencia, data_abertura, saldo, cancelado, senha
Métodos: abrir, encerrar, depositar, debitar, obter_saldo

Observe que algumas classes se relacionam com outras, como é o caso de pessoa com contas, através da propriedade cpf e cpf_titular.

Dica: Observe também que cada classe deve lidar com apenas um assunto.

24) Conceitos OO

Classe – uma classe é um template para a criação de objetos. A classe contém o código que define como os objetos devem se comportar e interagir com os demais objetos ou consigo mesmo. Cada vez que criamos um objeto de uma classe e herdando tudo que foi planejado na classe.
Objeto – um objeto é criado quando instanciamos uma classe. A classe é apenas o modelo, o que usamos pra valer são os objetos.
Instanciar - é o ato de criar um objeto de uma classe. Neste momento chama-se o construtor da classe instanciada. É executado assim: $objeto = new NomeClasse();
Propriedade – são similares às variáveis, mas pertencem a uma classe. O PHP não checa o tipo das propriedades.
Método – são semelhantes às funções mas pertencem a uma classe.
Membros – Membros de uma classe são suas propriedades e métodos.
Construtor – é o método da classe que é automaticamente executado sempre que se instancia uma classe.
Destrutor – é o método que quando executado destrói o objeto, invocado pela palavra-chave __destruct(). Será invocado automaticamente sempre ao final da execução do script.
Herança – O processo chave de derivar um objeto de uma classe é chamado de herança. A classe que extende a outra é chamada de subclasse e deriva todas as propriedades e métodos da superclasse (classe extendida). A subclasse então poderá processar cada método da superclasse.
extends – palavra reservada usada quando uma classe herda de outra, diz-se que ela extendeu a outra, ou seja ClasseSub extends ClasseSuper.
Interface -
Abstrata
Encapsulamento - Encapsulamento é o mecanismo que liga o código aos dados que ele manipula e mantém tanto a salvo de interferência externa e uso indevido. O fechamento de dados e métodos em uma única unidade (chamada classe) é conhecido como encapsulamento. O benefício do encapsulamento é que executa tarefas no interior sem preocupações.
Acoplamento – É a dependência e interdependência entre classes. Quanto menos acoplamento melhor.
Polimorfismo – É o princípio que permite que classes derivadas de uma mesma superclasse tenham métodos iguais (mesma assinatura e parâmetros) mas com comportamentos diferentes definidos em cada uma das classes . Este processo é chamado de polimorfismo. É o processo de criar vários objetos de uma classe básica específica. .
Setter -
Getter
this – this significa uma referência para a atual instância deste objeto. O this tanto dá acesso às propriedades quanto aos métodos. $this->body e $this->setSubject(). O this somente é válido no escopo de métodos que não sejam static.
self - usado para acessar membros static e const de dentro da classe. Para as demais usa-se this.
:: - operador usado para acessar membros static (inclusive const) no lugar de ->.
-> - operador utilizado para acessar membros de uma classe, exceto static.
Sobrescrever/Overwriting - Em um objeto derivado podemos sobrescrever qualquer um dos métodos herdados, desde que sejam
declarados como protected ou public e executem alguma coisa como desejamos. Simplesmente crie um método com o mesmo nome do que deseja sobrescrever.
Exemplo:
A classe SendMail tem um método enviarEmail(). O objeto $objmail criado desta classe herdará

este método. Queremos que o método tenha um comportamento diferente então criamos um novo método no objeto com o mesmo nome do objeto da classe pai, o que significa sobrescrever o método da classe pai.
Sobrecarregar/Overloading - Tanto chamada de métodos e acesso a membros podem ser sobrecarregados pelos
métodos __call, __get e __set. Esses métodos só serão disparados quando seu objeto ou o objeto herdado não contiver o membro public ou método que você está tentando acessar. Todos os métodos sobrecarregados devem ser definidos estáticos. Todos os métodos sobrecarregados devem ser definidos public.
A partir do PHP 5.1.0 também é possível sobrecarregar as funções isset() and unset() através dos métodos __isset e __unset respectivamente. O método __isset também é chamado com a função empty().
Sobrecarga de membros
void __set (string $name , mixed $value)
mixed __get (string $name)
bool __isset (string $name)
void __unset (string $name)
Membros de classes podem ser sobrecarregados para executar código específico definido na sua classe definindo esses métodos especialmente nomeados. O parâmetro $name usado é o nome da variável que deve ser configurada ou recuperada. O parâmetro $value do método __set() especifica o valor que o objeto deve atribuir à variável $name.
Nota: O método __set() não pode obter argumentos por referência.
implements – palavra-chave usada quando uma nova classe vai implementar uma interface.
new - É utilizada para criar novos objetos de uma classe. $obj = new Pessoa():
Os parêntesis ao final do nome da classe "Pessoa()" são opcionais, mas recomendados para tornar o código mais claro.

25) Padrões de Projeto (Design Patterns)

Inventados pela Gangue dos Quatro (Gang of Four – GoF). Foram criados para solucionar conjuntos de problemas similares de forma inteligente. Os padrões de projeto podem incrementar a performance dos aplicativos com código mínimo. Algumas vezes não é possível aplicar os design patterns. Desnecessário e não planejado uso dos DP pode também degradar a performance de aplicativos.

Padrões GoF
Os padrões "GoF" são organizados em famílias de padrões: de criação, estruturais e comportamentais. Os padrões de criação são relacionados à criação de objetos, os estruturais tratam das associações entre classes e objetos e os comportamentais das interações e divisões de responsabilidades entre as classes ou objetos.

Um padrão "GoF" também é classificado segundo o seu escopo: de classe ou de objeto. Nos padrões com escopo de classe os relacionamentos que definem este padrão são definidos através de herança e em tempo de compilação. Nos padrões com escopo de objeto o padrão é encontrado no relacionamento entre os objetos definidos em tempo de execução.

Padrões "GoF" organizados nas suas famílias:

Padrões de criação

- *Abstract Factory*
- *Builder*
- *Factory Method*
- *Prototype*
- *Singleton*

Padrões estruturais

- *Adapter*
- *Bridge*
- *Composite*
- *Decorator*
- *Façade*
- *Flyweight*
- *Proxy*

Padrões comportamentais

- *Chain of Responsibility*
- *Command*
- *Interpreter*
- *Iterator*
- *Mediator*
- *Memento*
- *Observer*
- *State*
- *Strategy*

- *Template Method*
- *Visitor*

Wikipédia - http://pt.wikipedia.org/wiki/Padr%C3%B5es_de_projeto_de_software

Apenas para citar alguns dos padrões de projeto.
O padrão de projeto mais comum hoje é o MVC.

26) Ferramentas para Trabalhar com PHP Orientado a Objetos

IDEs para programação com PHP

Eclipse PDT
http://www.eclipse.org/pdt/
http://www.eclipse.org/pdt/articles/debugger/os-php-eclipse-pdt-debug-pdf.pdf

NetBeans for PHP
http://www.netbeans.org/downloads/index.html

Ferramentas para trabalhar com UML em PHP

Umbrello - gera código PHP - for Linux (KDE) - http://uml.sourceforge.net/

ArgoUML - http://argouml.tigris.org/

DIA - http://projects.gnome.org/dia/

StartUML - http://staruml.sourceforge.net/en/

UML na Wikipedia - http://en.wikipedia.org/wiki/List_of_UML_tools

Tutorial Creating Use Case Diagrams - http://www.developer.com/design/article.php/2109801

Poseidon - http://www.gentleware.com/products.html
Versão para a comunidade:
* Download the software. - http://www.gentleware.com/downloadcenter.html

Documentação das classes com o PHP Documentor - http://www.phpdoc.org/
O PHP Documentor ou phpdoc é uma ferramenta de geração automática de documentação para código PHP documentado de forma adequada. Ele é similar ao JavaDoc.

Veja no diretório Documentacao como usar o PHP Documentor para documentar as classes em PHP.

27) Referências

Livro Object-Oriented Programming with PHP5
Autor - Hasin Hayder
Editora – Packt Publishing - http://www.packtpub.com
Código de Exemplo - http://www.packtpub.com/support/

E-book PHP 5 Power Programming
Autores - Andi Gutmans, Stig Sæther Bakken, and Derick Rethans
Editora - PRENTICE HALL – http://www.phptr.com

Livro Programando com Orientação a Objetos
Capítulo 1 de desmonstração
Autor - Pablo Dall'Oglio
Editora – Novatec – http://www.novatec.com.br

Aplicativo em PHPOO para a criação de um Blog
Abaixo um ótimo exemplo de aplicação simples criada com PHPOO e bem comentada, inclusive com código fonte para download:
http://net.tutsplus.com/news/how-to-create-an-object-oriented-blog-using-php/
Fontes: http://nettuts.s3.amazonaws.com/097_PHPBlog/SimpleBlogFiles.zip

Bom Tutorial sobre PHP Orientado a Objetos
http://www.phpro.org/tutorials/Object-Oriented-Programming-with-PHP.html