

DESENVOLVIMENTO AVANÇADO EM PHP

Sumário

DESENVOLVIMENTO AVANÇADO EM PHP	1
1. PHP e OO.....	6
1.1 Orientação a Objetos.....	6
1.1.1 Objeto.....	7
1.1.2 Classe.....	7
1.1.3 Atributo.....	7
1.1.4 Método.....	7
1.1.5 Herança.....	8
1.1.6 Polimorfismo.....	9
1.1.7 Acoplamento.....	9
1.1.8 Coesão.....	10
2. Orientação a Objetos aplicada a PHP	11
2.1 Classe.....	11
2.2 Pegando a dica: tipos de objetos.....	12
2.3 Visibilidade de Atributos e Métodos.....	14
2.3.1 Public.....	14
2.3.2 Protected.....	14
2.3.3 Private.....	14
2.4 Atributos e Métodos Estáticos.....	14
2.5 Constantes de Classes.....	15
2.6 Construtores e Destrutores.....	16
2.7 Herança.....	17
2.8 Métodos Finais.....	19
2.9 Classes Finais.....	20
2.10 Método __toString().....	20
3. OO avançada em PHP.....	22
3.1 Classes Abstratas.....	22
3.2 Interfaces.....	23
3.2.1 Vantagens da utilização de interfaces.....	24
3.2.2 Quando utilizar interfaces?.....	24
3.3 Interceptadores.....	25
3.3.1 __get().....	25
3.3.2 __set().....	26
3.3.3 __call().....	27
3.4 Reflexão.....	29
4. Controle de Exceções.....	31

4.1 Throw/Exception.....	31
4.2 Try/Catch.....	33
5. Funções para Manipulação de Classes.....	37
6. Introdução ao PEAR.....	39
6.1 Instalação.....	40
6.2 Usando o PEAR.....	40
6.3 Parâmetros de configuração.....	42
6.4 Comandos PEAR.....	43
6.4.1 pear install.....	43
6.4.2 pear list.....	44
6.4.3 pear info.....	45
6.4.4 pear list-all.....	46
6.4.5 pear list-upgrades.....	46
6.4.6 pear upgrade.....	46
6.4.7 pear upgrade-all.....	47
6.4.8 pear uninstall.....	47
6.4.9 pear search.....	47
6.4.10 pear remote-list.....	47
6.4.11 pear remote-info.....	47
6.4.12 pear download.....	48
6.4.13 pear config-get.....	48
6.4.14 pear config-set.....	48
6.4.15 pear config-show.....	48
6.4.16 pear shortcuts.....	48
7. Abstração de Banco de Dados com PEAR-MDB2.....	49
7.1 Instalação.....	49
7.2 DSN.....	50
7.3 Conectando.....	51
7.3.1 Limitando linhas ou lendo a partir de um offset.....	54
7.4 Resultados.....	55
7.4.1 Formato das linhas recuperadas.....	55
8. Scripts Shell PHP.....	57
8.1 Entrada do usuário.....	57
8.2 Analisando opções de linha de comando.....	57
8.3 Práticas Indicadas.....	59
8.3.1 Mensagem de uso.....	59
8.3.2 Código de Saída.....	60
8.3.3 Mensagens de Erro.....	61

9. Smarty.....	62
9.1 O que é o Smarty?.....	62
9.2 Instalação.....	64
9.3 Testando.....	64
9.4 Estrutura da aplicação.....	65
9.5 Informações sobre o layout.....	66
9.5.1 Delimitadores.....	66
9.5.2 Comentários.....	66
9.5.3 Funções.....	66
9.5.4 Atributos.....	67
9.5.5 Variáveis.....	67
9.5.6 Modificadores de Variáveis.....	68
9.5.7 Arquivos de Configuração.....	68
9.6 Informações sobre a programação PHP.....	69
9.6.1 A constante SMARTY_DIR.....	69
9.6.2 Variáveis de configuração.....	69
9.6.3 Métodos.....	70
9.7 Exemplo de uma aplicação com Smarty.....	70
10. Arquivos e Streams.....	76
10.1 Introdução.....	76
10.2 Acesso a arquivos.....	76
10.3 Input/Output de Programas.....	78
10.3.1 popen().....	78
10.3.2 proc_open().....	78
10.3.2.1 Descritores de Arquivos.....	79
10.3.2.2 Pipes.....	80
10.3.2.3 Arquivos.....	80
10.4 Streams de Input/Output.....	81
Dumping \$_POST.....	82
Dumping php://input.....	82
10.5 Streams de Compressão.....	83
10.6 Streams URL.....	84
11. Manipulação de Gráficos com GD.....	86
11.1 Formulários de submissão à prova de Bots.....	86
11.2 Gráfico de Barras.....	90
12. Expressões Regulares.....	96
12.1 Sintaxe.....	96

12.1.1 Sintaxe dos Padrões.....	97
12.1.2 Metacaracteres.....	97
12.2 Exemplo.....	100
12.3 Funções.....	101
12.3.1 Funções de combinação.....	101
12.3.2 Funções de substituição.....	102
Apêndice A - IDEs.....	104
O que é um IDE?.....	104
EasyEclipse.....	104
A interface do EasyEclipse.....	106

1. PHP e OO

O PHP 3 foi a versão que introduziu o suporte à programação orientada a objetos, embora utilizável, o suporte era extremamente rudimentar e não foi melhorado com o lançamento do PHP 4, no qual o desempenho, a modularidade e a compatibilidade reversa eram as maiores preocupações. Devido a uma forte demanda popular por um suporte a OO aprimorado, o modelo de objetos foi totalmente reelaborado na versão 5.

Uma das principais mudanças, se não a maior, trazida pela versão 5 do PHP é a nova maneira de tratar objetos, a partir desta versão o PHP trata objetos da maneira que deveriam ser tratados, não mais tratando objetos como tipos primitivos (integer, string, array, ...). Para programadores iniciantes pode parecer sem importância, mas a principal diferença é a que mais possui consequências, é o fato do PHP, a partir da versão 5, efetuar a atribuição de um tipo objeto a uma variável sem gerar uma cópia do mesmo, como era feito até então, mas a variável irá referenciar o objeto através de um ponteiro.

Neste capítulo abordaremos o conceito de Orientação a Objeto, um conceito comum para várias linguagens de programação, trataremos também da sintaxe Orientada a Objeto no PHP, e as principais novidades introduzidas com a nova versão da linguagem.

1.1 Orientação a Objetos

A orientação a objetos é o desenvolvimento de uma estratégia em que os sistemas devem ser construídos baseados em uma coleção de componentes reutilizáveis, conhecidos como objetos. Inicialmente, tenha em mente que objetos são entidades que conseguem reunir na mesma estrutura, dados e funções, de forma a estarem estritamente relacionados. Chamaremos os dados de um objeto de atributos, e suas funções de métodos, as quais podem manipular seus dados ou executar alguma ação relacionado ao objeto sem obrigatoriamente modificar seus atributos.

Tome como exemplo um robô. Um robô possui atributos (dados), como dimensões, peso, quantidade e tipo de sensores, posição atual, etc. E um robô também executa ações (métodos), dependendo do tipo de robô ele pode andar, falar, deslocar objetos, soldar, cortar, etc.

Antes de prosseguir, precisamos apresentar alguns conceitos que são fundamentais para o restante do capítulo:

1.1.1 Objeto

É a entidade que desejamos generalizar (torná-la uma classe) e pode ser uma pessoa, um lugar, um evento, um conceito, um relatório, uma tela, enfim qualquer coisa real. Pensando em termos de banco de dados, podemos enxergar um objeto como uma das linhas de uma tabela (por exemplo, um registro da tabela de clientes, de produtos, ou mesmo de condições de pagamento), mas lembre-se de que em OO esse objeto deve ter, além de dados, funcionalidades relacionadas a ele.

1.1.2 Classe

Uma classe é a abstração de um objeto, ou seja, é um modelo a partir do qual os objetos podem ser criados. Enquanto um objeto representa uma entidade real existente em determinado instante, uma classe é a generalização deste, mostrando quais devem ser suas características (dados) e quais suas capacidades (funcionalidades), poderíamos imaginar uma classe como uma “forma”, a qual servirá de molde para vários objetos! Tome, por exemplo, a classe funcionário. Ela deve descrever a entidade funcionário a qual deve ter características próprias, tais como: nome, código, função, salário e deve ter capacidades (funcionalidades), mas em termos de objeto pensaríamos não na classe funcionário, mas em um funcionário específico, por exemplo, o funcionário Rodrigo, o qual possui um nome, um código, uma função, um salário, etc.

1.1.3 Atributo

São os dados que uma classe deve possuir, isto é, as variáveis presentes dentro da classe, manipuladas por ela e que serão utilizadas apenas pela classe.

1.1.4 Método

Os métodos definem que as classes sabem fazer, ou seja, define a forma com que a classe irá manipular seus atributos, receber novas informações para serem armazenadas em seus atributos e também recuperar esses atributos, retornando-os para quem os chamou.

Com os conceitos apresentados até agora podemos ter uma idéia de como seria a estrutura de uma classe em PHP, ou seja, como definir seus atributos e métodos:

```
<?php
class funcionario {
    public $_codigo;
    public $_nome;
    public $_salario;
    public $_departamento;

    function getSalario() {
        return $this->_salario;
    }

    function setSalario($nsal) {
        $this->_salario = $nsal;
    }
}
?>
```

No exemplo acima temos os atributos \$_codigo, \$_nome, \$_salario, \$_departamento e os métodos getSalario() e setSalario(), para retornar o valor do salário e definir o valor de um novo salário respectivamente.

1.1.5 Herança

No desenvolvimento de sistemas, dos mais simples aos mais complexos, precisamos freqüentemente construir classes que muitas vezes são similares a outras já existentes, as quais compartilham parte de suas definições com atributos e métodos. E seria realmente interessante se pudéssemos reaproveitar os atributos e métodos de uma classe em outra, alterando somente o que é particular da nova classe. Bem, o conceito de herança é justamente isso, ou seja, é a capacidade de uma classe em herdar os atributos e métodos de uma outra classe (chamamos de superclasse a classe mãe, aquela que fornece os atributos e métodos à subclasse que os herda).

Tome como exemplo as entidades cliente e fornecedor, as quais têm os atributos código, nome, endereço, CNPJ, telefone, contato e alguns métodos em comum, logo podemos criar uma classe genérica chamada parceiros e fazer com que as classes cliente e fornecedor herdem os atributos e métodos da classe parceiro, realizando as alterações inerentes a cada uma.

A grande vantagem de utilização de herança está na reusabilidade dos códigos escritos por nós, pois se não tivermos esta possibilidade, precisaremos alterar a mesma coisa em vários pontos diferentes do sistema (o que acontece se temos duas classe, clientes e fornecedor, com códigos totalmente separados, cada uma fazendo sua validade de CNPJ e precisamos alterar essa validação? É preciso mexer nas duas classes, o que evidentemente é fácil se temos duas, mas e se temos 10, 20 ou mais classes que precisamos alterar?).

1.1.6 Polimorfismo

É a capacidade de dois ou mais objetos responderem à mesma mensagem de diferentes maneiras. Em termos de classes podemos pensar em superclasses e subclasses, em que o mesmo método possui comportamentos e resultados diferentes na superclasse e nas subclasses, desta forma quem chama o método não precisa distinguir qual deve chamar.

1.1.7 Acoplamento

É a medida do grau de ligação entre uma classe e outras existentes, ou seja, quão fortemente uma classe está conectada, depende ou conhece outras classes. Uma classe dita com fraco acoplamento não depende ou depende de poucas classes, o que representa a situação ideal na construção de sistemas, pois uma classe com forte acoplamento possui os seguintes problemas:

- Entendimento prejudicado, pois para conhecer o real comportamento de uma classe, precisamos conhecer e entender as classes às quais ela está acoplada.
- As alterações são em geral mais complexas, pois qualquer alteração nas classes acopladas pode afetar (muitas vezes não sabemos como) a classe.
- A reusabilidade é prejudicada, uma vez que a classe depende de outras, impedindo que somente a classe seja reusada. É preciso sempre carregar as demais classes.

Na vida real nem sempre podemos, ou conseguimos, excluir totalmente o acoplamento, mas podemos pelo menos minimizá-lo, reduzindo-o ao mínimo necessário e procurando realizar acoplamento apenas de dados (saída de um objeto é entrada de outro, utilização de parâmetros entre métodos dos objetos) ou no máximo de controles (Utilização de parâmetros de controle para determinar o comportamento de um objeto por outro).

1.1.8 Coesão

Mede o grau que uma classe ou seus métodos fazem sentido, ou seja, o quão claro é o entendimento do que a classe ou método faz. Uma forma eficaz de definir a coesão é tentarmos descrever a classe ou método em uma sentença. Quanto maior for a sentença menos coesa será a classe ou o método.

Um método é altamente coeso se realiza uma e somente uma função, por exemplo temos na classe funcionário, um método único para alterar todos os atributos da classe e ainda recalcular o salário conforme um percentual informado. Esse método (`alteraTudo()`) não é coeso, pois mistura várias funções. Deveríamos ter na verdade métodos separados para alterar cada um dos atributos da classe (`setNome`, `setSalario`, etc) e um método separado para calcular o novo salário (`calculaAumento`, por exemplo).

Uma classe altamente coesa representa e manipula um e somente um tipo de objeto. Uma classe que é de baixa coesão traz os seguintes problemas:

- É de difícil entendimento.
- Sua reusabilidade fica prejudicada.
- É de difícil manutenção.
- Qualquer mudança afeta vários pontos da classe, tornando-a complexa de lidar.
- Possui mais responsabilidades do que deveria, uma vez que assume funções que são de outras classes (incluir um novo departamento na classe de funcionários, por exemplo).

2. Orientação a Objetos aplicada a PHP

A partir da versão 5 do PHP temos disponíveis recursos existentes em outras linguagens fortemente orientadas a objetos como o Java. Recursos como gerenciamento de exceções (try, catch), herança, visibilidade de atributos e métodos, construtores, destrutores e reflexão, além de outros que veremos no decorrer deste capítulo.

2.1 Classe

Uma classe no PHP pode ser definida da seguinte forma:

```
<?php
class carro {
    private $_marca;
    private $_modelo;
    private $_cor;
    private $_ano;

    public function setMarca($_m) {
        $this->_marca = $_m;
    }

    public function setModelo($_m)    {
        $this->_modelo = $_m;
    }

    public function setCor($_c)  {
        $this->_cor = $_c;
    }

    public function setAno($_a)  {
        if(is_int($_a))  {
            $this->_ano = $_a;
        }
        else {
            return FALSE;
        }
    }

    public function getMarca()    {
        return $this->_marca;
    }

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

    public function getCor()      {
        return $this->_cor;
    }
}
```

```
public function getAno()      {
    return $this->_ano;
}

public function getCarro()    {
    return "Marca: " . $this->getMarca() . "<br/>" .
           "Modelo: " . $this->getModelo() . "<br/>" .
           "Cor: " . $this->getCor() . "<br/>" .
           "Ano: " . $this->getAno();
}
}
?>
```

Neste exemplo definimos a classe carro com três atributos: Marca, Modelo, Cor e Ano e os métodos para modificar estes atributos e recuperar seus valores, além de um método que retorna todos os atributos do objeto.

Note que dentro dos métodos da classe utilizamos a variável `$this` para referenciar a própria classe, e este deve ser o procedimento quando queremos referenciar a própria classe com seus métodos e/ou atributos.

Para instanciar (criar um objeto a partir de uma classe existente) uma classe no PHP, devemos utilizar a instrução *new* seguida do nome da classe e se for necessário, os parâmetros exigidos no método de construção da classe.

Por exemplo, vamos instanciar a classe carro recém criada:

```
<?php
$gol = new carro;

$gol->setMarca("Wolkswagen");
$gol->setModelo("Gol G4 Power");
$gol->setCor("Chumbo");
$gol->setAno(2008);

echo $gol->getCarro();
?>
```

Uma vez instanciada, podemos acessar os atributos e métodos públicos da classe. Para isso devemos utilizar o operador `->` após o nome do objeto que instância a classe (no exemplo anterior `$gol->`).

2.2 Pegando a dica: tipos de objetos

Assim como toda variável de argumento pode conter qualquer tipo primitivo, por padrão ela pode conter um objeto de qualquer tipo. Essa flexibilidade possui seus

usos, mas pode apresentar problemas no contexto de uma definição de método. Imagine um método projetado para trabalhar com um objeto carro:

```
class CarWrite {  
    public function write( $car ) {  
        $str = "$car->_marca <br/>";  
        $str .= "$car->_modelo<br/>";  
        $str .= "$car->_cor <br/>";  
        $str .= "$car->_ano <br/>";  
  
        print $str;  
    }  
}
```

Podemos testar a classe CarWriter da seguinte forma:

```
$writer = new CarWrite();  
$writer->write($gol);
```

A classe CarWrite contém um único método: write(). O método write() recebe um objeto carro, mas sem impor.

Para abordar o problema, o PHP 5 introduziu dicas de tipo de classe. Para adicionar uma dica de tipo a um argumento de método, você simplesmente coloca um nome de classe na frente do argumento do método que precisa restringir. Assim, podemos reformular nosso método write()

```
public function write( carro $car ) {  
    //...  
}
```

Agora, o método write() só receberá o argumento \$car se ele contiver um objeto do tipo carro. Experimente chamar write() com um objeto diferente.

Isso nos poupa de ter de testar o tipo do argumento antes de trabalhar com ele. Também torna a assinatura do método muito mais clara para o codificador cliente. Ele pode ver os requisitos do método write() com um rápido exame. Ele não tem de se preocupar com alguma falha obscura surgindo de um erro de tipo, pois a dica é imposta rigidamente.

Embora a verificação de tipo automatizada seja uma ótima forma de evitar falhas, é importante entender que dicas são verificadas em tempo de execução. Isso significa que uma dica de classe só reportará um erro no momento em que um objeto indesejado for passado para o método. Se uma chamada para o método write() for

enterrada em uma cláusula condicional que só é executada na manhã de natal, você pode se descobrir trabalhando no feriado se não tiver verificado seu código com cuidado.

2.3 Visibilidade de Atributos e Métodos

Os métodos e atributos de uma classe no PHP podem ser definidos como: `private` (privativos), `public` (públicos) ou `protected` (protegidos), adicionando uma dessas palavras chaves antes do nome do atributo ou da palavra chave `function` na definição de um método. Cada um dos tipos define como o atributo ou método se comporta perante a classe, subclasses e o restante do sistema. Suas definições são:

2.3.1 *Public*

Um atributo ou método definido como `public`, torna-se acessível de qualquer parte da classe, de suas subclasses, bem como em qualquer parte dos scripts que contêm a classe.

2.3.2 *Protected*

Atributos ou métodos definidos como `protected` são visíveis pela classe que os criou e por suas subclasses (classes que herdam a classe principal), porém não são acessíveis fora deste contexto (fora da classe principal ou suas subclasses, como, por exemplo, no script que conte a definição da classe).

2.3.3 *Private*

Os atributos e métodos definidos como `private` são visíveis apenas na classe que os criou, ou seja, subclasses ou o script que contêm a classe não podem acessar esses atributos ou métodos.

Nota: para manter a compatibilidade com sistemas escritos com PHP4 ou 3, o PHP5 aceita na definição de classes o termo `var`, assumindo `public` em seu lugar, porém uma mensagem de erro do tipo `E_STRICT` será gerada.

2.4 Atributos e Métodos Estáticos

Um atributo ou método definido como estático, através da palavra reservada `static` logo após a definição de visibilidade do atributo ou do método, faz com que o

atributo ou método em questão, pertença estritamente a classe, e não aos objetos baseados na classe. Portanto são freqüentemente chamados de propriedades de classe.

Os atributos e métodos estáticos não são acessíveis através de uma instância da classe (da variável que instanciou a classe), e também não podem ser redefinidos nas subclasses, desta forma a variável especial `$this` não está disponível dentro dos métodos definidos como estáticos, devemos utilizar `self` em seu lugar. Em outras palavras, se estiver acessando propriedades ou métodos estáticos a partir de um script que contém a definição da classe, deve-se utilizar o nome da classe seguido do operador "::", por exemplo: `myclass::mystaticmethod()`. Caso esteja acessando um atributo ou método estático de uma classe a partir de um objeto da mesma classe, deve-se utilizar a palavra reservada **self**, por exemplo: `self::mystaticmethod()`.

```
<?php
class MyUniqueIdClass {
    static $idCounter = 0;

    public $uniqueId;

    function __construct()
    {
        self::$idCounter++;
        $this->uniqueId = self::$idCounter;
    }
}

$obj1 = new MyUniqueIdClass();
print $obj1->uniqueId . "<br/>";

$obj2 = new MyUniqueIdClass();
print $obj2->uniqueId . "<br/>";

?>
```

O exemplo acima atribui um id singular a todas as instâncias de uma classe.

2.5 Constantes de Classes

Com o aprimorado suporte ao encapsulamento do PHP 5, agora podemos definir constantes dentro de classes. Similares aos membros estáticos, elas pertencem à classe, e não as instâncias da classe. As constantes de classes sempre distinguem

entre caixa alta e baixa. A sintaxe da declaração é intuitiva, e o acesso às constantes se faz de forma semelhante ao acesso a membros estáticos:

```
<?php
class MyColorEnumClass {
    const RED = "Red";
    const GREEN = "Green";
    const BLUE = "Blue";

    function printBlue() {
        print self::BLUE;
    }
}

print MyColorEnumClass::RED;
$obj = new MyColorEnumClass();
$obj->printBlue();
?>
```

Este código exibe “Red” seguido por “Blue”. Ele demonstra a habilidade de se acessar a constante tanto de dentro do método de uma classe, com a palavra chave `self`, como a partir do nome da classe “`MyColorEnumClass`”. Como o nome já diz, as constantes são constantes, e não podem ser modificadas nem removidas após serem definidas. Usos comuns para constantes incluem definição de enumerações, tais como no exemplo anterior, ou algum valor de configuração, como o nome de usuário do banco de dados, que você não gostaria que a aplicação pudesse modificar.

Nota: escrever nomes de constantes de classes com caixa alta é um consenso entre os programadores, apesar de não ser obrigatório.

2.6 Construtores e Destrutores

Construtores referenciados no PHP 5 como `__construct()`, é uma função definida na classe e que é executada sempre que o objeto é criado (isto é, sempre que a classe é instanciada). O destruidor da classe, que é definido através da função `__destruct()`, é executado sempre que o objeto for destruído, explícita ou implicitamente (por exemplo, no término do script que utilizava a classe).

A função `__construct()` pode ser construída de forma que aceite parâmetros na sua definição, já a função `__destruct()` não aceita parâmetros.

Nota: para manter a compatibilidade o PHP 5, em se tratando de construtores, procura primeiramente pelo método `__construct()`, caso não encontre este método,

procura por uma função com o mesmo nome da classe. No caso de destrutores, este conceito não existia antes do PHP 5.

Nota: Classes que estendem outras classes, ou seja, subclasses, não executam automaticamente o construtor da superclasse. O método precisa ser invocado explicitamente através de `parent::__construct()`. O mesmo vale para a função `__destruct()`.

2.7 Herança

No PHP 5 uma classe pode herdar atributos ou métodos de uma outra classe. Basta acrescentar após o nome da classe, na sua definição, a palavra reservada **extends** seguida do nome da classe da qual esta vai herdar atributos e métodos, desta forma a classe que cede os atributos e métodos é chamada de superclasse e a que recebe os atributos e métodos (ou seja, a herdeira) recebe o nome de subclasse.

A subclasse pode utilizar os atributos e métodos existentes na superclasse, sobreescrever os métodos da superclasse ou criar outros atributos ou métodos. O PHP procura primeiramente o método na subclasse e caso o encontre, ele será executado. Caso o método não exista na subclasse, ele será procurado na superclasse. A exceção são os métodos definidos com a palavra reservada **final**, indicando que as subclasses não podem sobreescrever esse método.

Para acessar os atributos e métodos da superclasse, devemos utilizar a palavra reservada **parent** seguida de `::` (ou seja, `parent::getValor()` por exemplo).

A seguir um exemplo que reúne os últimos conceitos abordados, com exceção de atributos e métodos estáticos, vistos no exemplo acima.

```
<?php
class carro {
    protected $_marca;
    protected $_modelo;
    protected $_cor;
    protected $_ano;

    function __construct($_mr="", $_md="", $_c="", $_a=0) {
        $this->setMarca($_mr);
        $this->setModelo($_md);
        $this->setCor($_c);
        $this->setAno($_a);
    }

    public function setMarca($_m) {
        $this->_marca = $_m;
    }
}
```

```
public function setModelo($_m)    {
    $this->_modelo = $_m;
}

public function setCor($_c)    {
    $this->_cor = $_c;
}

public function setAno($_a)    {
    if(is_int($_a))    {
        $this->_ano = $_a;
    }
    else {
        return FALSE;
    }
}

public function getMarca()    {
    return $this->_marca;
}

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

public function getCor()    {
    return $this->_cor;
}

public function getAno()    {
    return $this->_ano;
}

public function getCarro()    {
    return "Marca: " . $this->getMarca() . "<br/>" .
           "Modelo: " . $this->getModelo() . "<br/>" .
           "Cor: " . $this->getCor() . "<br/>" .
           "Ano: " . $this->getAno();
}

function __destruct() {
    echo $this->getCarro();
    echo "__FIM (Carro)__<br/>";
}
}

class caminhao extends carro {
    protected $_eixos;

    function __construct($mr="", $md="", $c="", $a=0, $ne=0) {
        parent::__construct($mr, $md, $c, $a);
        $this->setEixo($ne);
    }

    public function setEixo($_ne) {
        $this->_eixos = $_ne;
    }
}
```

```
}

public function getEixo() {
    return $this->_eixos;
}

public function getCarro() {
    return "Dados do Caminhão:<br/>" .
        "Marca: " . $this->getMarca() . "<br/>" .
        "Modelo: " . $this->getModelo() . "<br/>" .
        "Eixos: " . $this->getEixo() . "<br/>" .
        "Cor: " . $this->getCor() . "<br/>" .
        "Ano: " . $this->getAno() . "<br/>";
}

function __destruct() {
    echo "__FIM (Caminhão)__<br/>";
}
}

$_c = new caminhao("MB", "1113", "Vermelho", 1998, 4);
$_c->setCor("VERDE");
echo $_c->getCarro();

?>
```

Note que todos os métodos disponíveis na classe **carro** também estão disponíveis na classe **caminhao**, e os métodos **__construct()**, **__destruct()** e **getCarro()** foram sobrescritos na classe **caminhao**, fazendo com que os métodos originais sejam desconsiderados na chamada ao método da subclasse (para que sejam executados devem ser chamados de forma explícita através de **parent::**). Veja ainda que o método **setCor** (que não possui um equivalente na subclasse **caminhao**) utilizou o método definido na superclasse **carro**.

Nota: não confunda, utilizamos a palavra reservada **parent** para referenciar atributos ou métodos (geralmente estáticos) e também constantes da **superclasse** a partir de uma subclasse, e utilizamos **self** para acessar atributos ou métodos (geralmente estáticos) e também constantes da **própria classe**.

2.8 Métodos Finais

Até agora vimos que, quando se estende uma classe (ou herda de uma classe), podemos substituir métodos herdados por uma nova implementação. Entretanto, há momentos em que poderíamos desejar que um método não fosse reimplementado em suas classes derivadas. Para esse propósito, o PHP suporta o

modificador de acesso **final**. O seguinte exemplo não é um script PHP válido porque está tentando substituir um método final:

```
<?php
class MyBaseClass {
    final function idGenerator()
    {
        return $this->id++;
    }

    protected $id = 0;
}

class MyConcreteClass extends MyBaseClass {
    function idGenerator()
    {
        return $this->id += 2;
    }
}
?>
```

2.9 Classes Finais

De modo semelhante aos métodos finais, podemos também definir uma classe como final. Isso desabilita a possibilidade de alguma classe derivada herdar a classe em questão. O seguinte código não funciona:

```
final class MyBaseClass {
    . . .
}

class MyConcreteClass extends MyBaseClass {
    . . .
}
```

2.10 Método __toString()

Considere o seguinte código:

```
class Person {

    private $name;

    function __construct($name)
    {
        $this->name = $name;
    }
}
```

```
$obj = new $Person("Kleber Silva");  
print $obj;
```

Isso irá exibir:

Object id #1

Diferentemente da maioria dos outros tipos de dados, exibir o id do objeto normalmente não será interessante para nós. Além disso, os objetos freqüentemente se referem a dados que deveriam ter semântica de exibição – por exemplo, poderia fazer sentido que, quando você exibisse um objeto de uma classe que representa uma pessoa, as informações da pessoa fossem exibidas conjuntamente. Para esse propósito, o PHP permite que você implemente uma função chamada `__toString()`, que deve retornar a rerepresentação em string do objeto e, quando definido, o comando `print` a chamará e exibirá a string retornada. Usando-se `__toString()`, o exemplo anterior pode ser modificado para a sua forma mais útil:

```
class Person {  
    private $name;  
  
    function __construct($name) {  
        $this->name = $name;  
    }  
  
    function __toString() {  
        return $this->name;  
    }  
}  
  
$obj = new $Person("Kleber Silva");  
print $obj;
```

Isso irá exibir:

Kleber Silva

Nota: o método `__toString()` só é chamado pelos construtores `print` e `echo` da linguagem. No futuro, eles provavelmente também poderão ser chamados pelas operações comuns de strings, tais como a concatenação e a conversão explícita para strings.

3. OO avançada em PHP

Neste capítulo iremos abordar os conceitos mais avançados do núcleo Orientado a Objeto introduzidos com a versão 5 do PHP. Ao final deste capítulo você deverá ter conhecimento do que são classes abstratas, interfaces, reflexão, controle de exceções além de algumas funções úteis para manipulação de classes.

3.1 Classes Abstratas

Ao elaborar hierarquias de classes, poderíamos desejar que certos métodos para as classes que recebem herança implementarem. Por exemplo, digamos que você tenha uma classe genérica para figuras geométricas chamada shape, e duas classes derivadas square e circle. Onde a classe shape possui o método setCenter e o atributo draw, que serão herdados pelas subclasses.

Faria muito sentido implementar setCenter() na classe shape e deixar a implementação do método draw() para as classes concretas square e circle, já que se desenharam de maneiras diferentes. Para isso deveríamos declarar o método draw() como um método abstrato, para que o PHP saiba que estamos deixando intencionalmente de implementá-lo na classe shape. Desta forma a classe shape seria uma classe abstrata, significando que não possui funcionalidade completa, e cuja função é servir de origem para heranças.

Não é possível instanciar classes abstratas. Você pode definir qualquer número de métodos como abstract, mas, uma vez definido como abstract pelo menos um método de uma classe, então toda a classe precisa ser declarada como abstract também. Essa dupla definição existe para dar-lhe a opção de definir uma classe como abstract mesmo se ela não possuir quaisquer métodos abstract, e para lhe forçar a definir como abstract uma classe com métodos abstract, de modo que fique claro para as pessoas o que você tinha em mente.

Para definir uma classe como abstrata, basta incluir a palavra reservada **abstract** antes de **class**, e para definir um método como abstrato, basta incluir a palavra **abstract** antes da definição do método (antes da visibilidade).

A visibilidade definida nas subclasses deve ser igual ou mais fraca que a definida no método da classe abstrata (por exemplo, se a classe abstrata definiu a visibilidade como protected, as subclasses devem defini-la como protected ou public, nunca como private).

Nota: qualquer classe que tenha pelo menos um método abstrato deve ser definida como abstrata também.

```
<?php
abstract class Shape {

    protected $x, $y;

    function setCenter($x, $y) {
        $this->x = $x;
        $this->y = $y;
    }

    abstract function draw();
}

class Square extends Shape {
    function draw() {
        echo "desenhando um quadrado...<br/>";
    }
}

class Circle extends Shape {
    function draw() {
        echo "desenhando um circulo...<br/>";
    }
}

$obj1 = new Square;
$obj1->draw();

$obj2 = new Circle;
$obj2->draw();

?>
```

3.2 Interfaces

Uma interface permite definir quais métodos públicos uma classe deve implementar, sem ser necessário definir como os métodos devem ser construídos.

A herança de classes lhe permite descrever uma relação mãe-filha entre classes, por exemplo, poderíamos ter uma classe base shape, da qual tanto square como circle derivam. Entretanto, freqüentemente poderíamos desejar adicionar outras “interfaces” às classes, basicamente significando funcionalidades adicionais que a classe deve ter. Isso é feito em algumas linguagens de programação usando-se herança múltipla e derivando a partir de duas classes. O PHP escolheu as interfaces

como uma alternativa à herança múltipla, o que lhe permite especificar funcionalidades adicionais que uma classe deve possuir.

As interfaces são definidas da mesma forma que as classes, com exceção que devem ter a palavra reservada `interface` no lugar de `class` e não podem implementar os métodos definidos. Elas devem apenas definir sua assinatura (visibilidade, nome e parâmetros) assim como em métodos abstratos.

As classes baseadas em uma ou mais interfaces devem incluir a palavra reservada `implements` e a lista de interfaces que vão implementar, separadas por vírgula. Todos os métodos definidos nas interfaces devem ser implementados; caso contrário, um erro fatal será gerado no PHP.

3.2.1 Vantagens da utilização de interfaces

Interfaces possibilitam que você defina estruturas comuns para suas classes, para definir um padrão para seus objetos.

Interfaces solucionam o problema de herança simples. Elas possibilitam que você adicione “qualidades” de múltiplas fontes.

Interfaces possibilitam uma base/raiz flexível, a qual você não consegue utilizando classes.

Interfaces são ótimas quando se tem múltiplos programadores trabalhando em um projeto. Você pode definir uma estrutura (modelo), para seus programadores seguirem, deixando que eles se preocupem com os detalhes.

3.2.2 Quando utilizar interfaces?

Se você possui uma classe que nunca será diretamente instanciada em seu programa, ela é uma boa candidata à uma interface. Em outras palavras, se você está criando uma classe apenas para servir de superclasse (classe mãe) para outras classes, isto provavelmente poderia ser implementado em uma interface.

Quando você sabe quais métodos uma classe deverá ter, mas não sabe exatamente como eles devem ser implementados.

Quando você deseja mapear as estruturas básicas para suas classes, para servir de template para suas classes seguirem, mantendo a base de código consistente.

3.3 Interceptadores

O PHP fornece métodos interceptadores internos, que podem interceptar mensagens enviadas para métodos e propriedades não definidas. Isso também é conhecido como “sobrecarga”, mas, já que esse termo significa algo bastante diferente em Java e C++, vamos falar em termos de interceptação.

O PHP 5 suporta três métodos internos de sobrecarga. Como o `__construct()`, eles são chamados quando as condições corretas são satisfeitas.

Método	Descrição
<code>__get(\$property)</code>	Chamado quando uma propriedade indefinida é acessada.
<code>__set(\$property, \$value)</code>	Chamado quando um valor é atribuído a uma propriedade indefinida.
<code>__call(\$method, \$arg_array)</code>	Chamado quando um método indefinido é habilitado.

Os métodos `__get()` e `__set()` são projetados para trabalhar com propriedades que não tiverem sido declaradas em uma classe (ou nas suas ancestrais).

3.3.1 `__get()`

O `__get()` é chamado quando o código cliente tenta ler uma propriedade não declarada. Ele é chamado automaticamente com um único argumento string contendo o nome da propriedade que o cliente estiver tentando acessar. O que quer que você retorne do método `__call()` será enviado para o cliente como se a propriedade alvo existisse com esse valor. Eis um exemplo:

```
class Person {  
    function __get($property) {  
        $method = "get{$property}";  
        if( method_exists( $this, $method) ) {  
            return $this->$method();  
        }  
    }  
  
    function getName() {  
        return "Bob";  
    }  
  
    function getAge() {  
        return 44;  
    }  
}
```

```
}  
}
```

Quando um cliente tenta acessar uma propriedade não definida, o método `__get()` é chamado. Implementamos `__get()` para pegar o nome da propriedade e construir uma nova string, pré-concebendo a palavra `get`. Passamos essa string para uma função chamada `method_exists()`, que recebe um objeto e um nome de método e testa se o método existe. Se existir, o chamamos e passamos seu valor de retorno para o cliente. Assim, se o cliente solicitar uma propriedade `$name`:

```
$p = new Person();  
print $p->name;
```

Saída

Bob

O método `getName()` é chamado em segundo plano. Se o método não existir, não fazemos nada. A propriedade que o usuário está tentando acessar será resolvida como `NULL`.

3.3.2 `__set()`

O método `__set()` é chamado quando o código cliente tenta atribuir a uma propriedade não definida. Ele recebe dois argumentos: o nome da propriedade e o valor que o cliente está tentando gravar. Você pode, então, decidir como trabalhar com esses argumentos. Reformularemos a classe `Person`:

```
class Person {  
    private $_name;  
    private $_age;  
  
    function __set($property, $value) {  
        $method = "set{$property}";  
        if( method_exists( $this, $method) ) {  
            return $this->$method($value);  
        }  
    }  
  
    function setName($name) {  
        $this->_name = strtoupper($name);  
    }  
  
    function setAge($age) {  
        $this->_age = strtoupper($age);  
    }  
}
```

```
}  
}
```

No exemplo, trabalhamos com métodos de gravação, em vez de leitura. Se um usuário tentar atribuir a uma propriedade não definida, o método `__set()` é chamado com o nome da propriedade e com o valor atribuído. Testamos a existência do método apropriado e o chamamos se ele existir. Dessa forma, podemos filtrar o valor atribuído.

Nota: lembre-se de que métodos e propriedades na documentação PHP são freqüentemente mencionados em termos estáticos para identificá-los com suas classes. Assim, poderíamos falar sobre a propriedade `Person::$name`, embora a propriedade não seja declarada como static e seja, na verdade, acessada por um objeto, da seguinte maneira:

```
$person->name;
```

Se criarmos um objeto `Person` e tentarmos gravar uma propriedade chamada `Person::$name`, o método `__set()` é chamado, pois esta classe não define uma propriedade `$name`. O método recebe uma string `name` e o valor que desejamos gravar. Depende de nós o que fazer com a informação. Nesse exemplo, construímos um nome de método do argumento propriedade combinado com a string `set`. O método `setName()` é encontrado e devidamente chamado. Ele transforma o valor de entrada e o armazena em uma propriedade real:

```
$p = new Person();  
$p->name = "bob";
```

A propriedade `$_name` se torna 'BOB'

3.3.3 `__call()`

O método `__call()` é, provavelmente, o mais útil de todos os métodos interceptadores. Ele é chamado quando um método não definido é habilitado pelo código cliente. O `__call()` é chamado com o nome do método e uma matriz que armazena todos os argumentos passados pelo cliente. Qualquer valor que você retorne do método `__call()` é retornado para o cliente como se tivesse sido feito pelo método chamado.

O método `__call` pode ser útil para a delegação. Delegação é o mecanismo por meio do qual um objeto passa chamadas de métodos. Ele é semelhante a herança no

sentido em que uma classe-filha passa uma chamada de método para sua implementação ancestral. Com herança, o relacionamento entre filha e mãe é fixo, então, o fato de que você pode alternar o objeto recebido em tempo de execução significa que a delegação pode ser mais flexível do que a herança. Clarearemos um pouco essa questão com o exemplo abaixo. Veja uma classe simples para formatar informações da classe Person:

```
class PersonWriter {  
    function writeName ( Person $p ) {  
        print $p->getName()."<br/>";  
    }  
  
    function writeAge( Person $p ) {  
        print $p->getAge()."<br/>";  
    }  
}
```

Poderíamos, é claro, criar subclasses para mostrar dados de Person de diversas formas. Veja a implementação da classe Person, que usa tanto um objeto PersonWriter quanto o método __call():

```
class Person {  
    private $writer;  
  
    function __construct ( PersonWriter $writer ) {  
        $this->writer = $writer;  
    }  
  
    function __call( $methodname, $args ) {  
        if( method_exists( $this->writer, $methodname ) ) {  
            return $this->writer->$methodname( $this );  
        }  
    }  
  
    function getName() { return "Bob"; }  
    function getAge() { return 44; }  
}
```

A classe Person demanda um objeto PersonWriter como argumento do construtor e o armazena em uma variável de propriedade. No método __call(), usamos o argumento \$methodname fornecido, testando um método do mesmo nome no objeto PersonWriter que armazenamos. Se encontrarmos tal método, delegamos a chamada do método ao objeto PersonWriter, passando nossa instância corrente para ele (na pseudo-variável \$this). Assim, o cliente faz a chamada a Person:

```
$person = new Person (new PersonWriter() );  
$person->writeName();
```

O método `__call()` é chamado. Encontramos um método chamado `writeName()` no seu objeto `PersonWriter` e o chamamos. Isso evita que precisemos chamar manualmente o método delegado:

```
function writeName() {  
    $this->writer->writeName( $this );  
}
```

Nota: a classe `Person` ganhou dois novos métodos. Embora a delegação automatizada possa economizar muito trabalho, se você tiver de delegar para muitos métodos, há um custo quanto à clareza. Você apresenta ao mundo uma interface dinâmica que resistirá à reflexão (o exame em tempo de execução de aspectos da classe) e não será clara para o codificador cliente à primeira vista. Os métodos interceptadores possuem seu lugar, mas devem ser usados com cautela. As classes que se baseiam neles devem documentar esse fato com muita clareza.

3.4 Reflexão

O PHP 5 disponibiliza um conjunto de classes que permite uma engenharia reversa completa em qualquer classe, interface, função, método, e adicionalmente esse conjunto (API) permite que sejam recuperados os comentários existentes em classes, funções e métodos.

A classe **reflection** possui o método estático **export**, o qual retorna a descrição completa da classe informada, a qual deve ser instanciada antes com **reflectionClass** da seguinte forma:

```
reflection::export(new reflectionclass('nome_classe'))
```

Experimente utilizar o código acima para alguma classe já utilizada nos exemplos anteriores.

A classe **reflectionFunction** permite que seja documentada qualquer função. Assim como a classe **reflection**, essa classe possui também uma função **export**, que deve ser invocada da seguinte maneira:

```
reflectionFunction::export('nome_funcao')
```

A classe **reflectionClass** possibilita a engenharia reversa em uma determinada classe. Para utilizarmos a classe, devemos primeiramente instanciá-la (a não ser que seja utilizado apenas o método **export**, neste caso precisamos somente realizar uma chamada ao método na forma **ReflectionClass::export**).

```
$_c = new reflectionClass('nome_classe');
```

ou:

```
echo reflectionClass::export('nome_classe');
```

A classe **ReflectionMethod** realiza a engenharia reversa em métodos de classes, a qual deve ser instanciada antes de ser utilizada, com exceção para o método **export** como nas outras classes.

```
echo reflectionMethod::export('nome_classe',  
'nome_metodo');
```

Podemos utilizar a o método estático **export** da classe **reflectionClass** para todas as classes de reflexão vistas:

```
echo reflectionClass::export('reflection');  
echo reflectionClass::export('reflectionClass');  
echo reflectionClass::export('reflectionFunction');  
echo reflectionClass::export('reflectionMethod');
```

Com as instruções acima podemos visualizar quais os métodos, de cada uma das classes e utilizar os métodos adequados em qualquer instância de alguma das classes acima.

4. Controle de Exceções

Na versão 5 do PHP temos mais controles para gerenciamento de erros, possibilitando um controle mais eficiente e limpo dos scripts, pois até a versão 4 precisávamos desenvolver nossos próprios controles, com vários pontos de validação e chamada a funções do tipo **trigger_error()**.

Agora podemos criar blocos do tipo try/catch, comuns em outras linguagens orientadas a objeto (tais como: Java e Object Pascal) e existe ainda a classe **exception** que disponibiliza um melhor controle de exceções geradas no script.

As funções de manipulação de erros disponíveis nas versões anteriores continuam válidas e possuem sua utilidade, porém agora temos mais ferramentas para um gerenciamento mais limpo e organizado dos erros e exceções.

A classe exception possui os seguintes métodos:

- **__construct**: No construtor da classe é obrigatória a definição de uma mensagem e opcionalmente um número de erro associado.
- **getCode**: O código do erro (se passado para a classe).
- **getFile**: Nome do arquivo no qual a exceção foi gerada.
- **getLine**: Número da linha em que a exceção foi gerada.
- **getTrace**: Um array com informações sobre cada etapa na progressão de uma exceção.
- **getTraceAsString**: O mesmo que getTrace(), só que em uma string em vez de um array.

4.1 Throw/Exception

Um exemplo para utilização da classe **exception**:

```
<?php
    $_e = new Exception("Não é possível dividir por zero", 1044);
?>
```

A classe **exception** é muito útil quando utilizada em conjunto com a palavra reservada **throw**, a qual encerra o método executado imediatamente e lança a exceção gerada para o sistema. A sintaxe de **throw** é:

throw exceção

Uma das formas mais utilizada para **throw** é:

```
Throw new exception(exceção);
```

```
<?php
class mostra_arquivo {
    protected $_nome_arq;
    private $_flg;

    function __construct($_nome) {
        $this->_flg = FALSE;
        $this->setArq($_nome);
    }

    public function setArq($_nome) {
        if(file_exists($_nome)) {
            $this->_nome_arq = $_nome;
            $this->_flg = TRUE;
        }
        else {
            throw new exception("Arquivo informado não existe",
4);
        }
    }

    public function mostra() {
        if($this->_flg === FALSE) {
            throw new exception("Arquivo não informado", 10);
        }
        elseif(file_exists($this->_nome_arq)) {
            if(filisize($this->_nome_arq)==0) {
                throw new exception("Arquivo vazio");
            }
            echo "<pre>";
            echo htmlentities(file_get_contents($this-
>_nome_arq));
            echo "</pre>";
        }
        else {
            throw new exception("Arquivo informado não existe",
40);
        }
    }
}
echo "<pre>";
$_a = new mostra_arquivo("Arquivo.txt");
echo "</pre>";

?>
```


Note que a exceção gerada com `throw` resulta em um erro fatal, garantindo que o código seja executado de forma correta, ou seja, que uma exceção gerada no script interrompa a sua execução, evitando desta forma que outras instruções sejam executadas erroneamente.

4.2 Try/Catch

Uma outra forma de gerenciamento de exceções é a utilização da estrutura **try/catch**. Essa estrutura consiste em uma cláusula **try** seguida de pelo menos uma cláusula **catch**.

```
try {  
    comandos  
} catch (class_exceção exceção) {  
    tratamento das exceções  
}
```

Qualquer comando dentro do bloco **try** que gerar uma exceção será enviado para o bloco **catch**.

```
<?php  
class mostra_arquivo {  
    protected $_nome_arq;  
    private $_flg;  
  
    function __construct($_nome) {  
        $this->_flg = FALSE;  
        $this->setArq($_nome);  
    }  
  
    public function setArq($_nome) {  
        if(file_exists($_nome)) {  
            $this->_nome_arq = $_nome;  
            $this->_flg = TRUE;  
        }  
        else {  
            throw new exception("Arquivo informado não existe",  
4);  
        }  
    }  
  
    public function mostra() {  
        if($this->_flg === FALSE) {  
            throw new exception("Arquivo não informado", 10);  
        }  
        elseif(file_exists($this->_nome_arq)) {  
            if(filesize($this->_nome_arq) == 0) {  
                throw new exception("Arquivo vazio");  
            }  
            echo "<pre>";  
        }  
    }  
}
```

```

        echo htmlentities(file_get_contents($this-
>_nome_arq));
        echo "</pre>";
    }
    else {
        throw new exception("Arquivo informado não existe",
40);
    }
}
}

function exhibe($_a) {
    try {
        $_a = new mostra_arquivo($_a);
        $_a->mostra();
    } catch (Exception $e) {
        echo "Erro: ". $e->getCode() . " - " . $e->getMessage();
        if($e->getCode()>10) {
            exit();
        }
    }
}

exib("Arquivo.txt"); //Erro
exib("mdb2.php"); //OK
?>

```

Como se nota, o objeto Exception é disponibilizado, na ocorrência de uma exceção (quando throw é invocado), para a cláusula catch, possibilitando que ele seja manipulado conforme nossa conveniência.

Podemos, ainda sofisticar a manipulação de exceções, definindo vários tipos diferentes de exceções, e conforme elas são geradas, podemos tratá-las da forma mais conveniente. Isso é conseguido por meio da extensão (herança) da classe Exception, quando catch for invocado (na ocorrência de uma exceção, podemos tratar as exceções conforme a classe que as gerou). O exemplo seguinte mostra como tratar uma exceção desta forma.

```

<?php

class EArquivoNaoExiste extends Exception {}
class EformatoArquivo extends Exception {}
class EconteudoArquivo extends Exception {}

class mostra_arquivo {
    protected $_nome_arq;
    private $_flg;

    function __construct($_nome) {
        $this->_flg = FALSE;
        $this->setArq($_nome);
    }
}

```

```

    }

    public function setArq($_nome) {
        if(file_exists($_nome)) {
            $this->_nome_arq = $_nome;
            $this->_flg = TRUE;
        }
        else {
            throw new EArquivoNaoExiste($_nome);
        }
    }

    public function mostra() {
        if($this->_flg === FALSE) {
            throw new exception("Arquivo não informado", 10);
        }
        elseif(file_exists($this->_nome_arq)) {
            if( is_executable($this->_nome_arq) or
                is_dir($this->_nome_arq)) {
                throw new EFormatoArquivo($this->_nome_arq);
            }
            if(filesize($this->_nome_arq) == 0) {
                throw new EConteudoArquivo("Arquivo vazio");
            }
            echo "<pre>";
            echo htmlentities(file_get_contents($this->_nome_arq));
            echo "</pre>";
        }
        else {
            throw new EArquivoNaoExiste($this->_nome_arq);
        }
    }
}

function exhibe($_a) {
    try {
        $_a = new mostra_arquivo($_a);
        $_a->mostra();
    } catch (EArquivoNaoExiste $e) {
        echo "(Erro) O Arquivo". $e->getMessage() . " não
existe!";
    } catch (EConteudoArquivo $e) {
        echo "(Erro) O Arquivo informado está " . $e->getMessage();
    } catch (EFormatoArquivo $e) {
        echo "(Erro) O Formato do Arquivo não é válido (" .
            $e->getMessage() . ")";
    } catch (Exception $e) {
        echo "ERRO Desconhecido: " . $e->getCode() . " - " . $e->getMessage();
        exit();
    }
}

exibe("Arquivo.txt"); //Erro
exibe("mdb2.php"); //OK

```

```
exibe("/bin/ls"); //Erro  
?>
```

Note que a exceção genérica `catch (Exception $e)` foi inserida como última cláusula `catch` porque, como todas as subclasses de erro são extensões de `Exception`, qualquer erro gerado pertencerá a classe `Exception`. Se a cláusula `catch (Exception $e)` for inserida logo após o bloco **try**, todas as exceções geradas serão tratadas nesse primeiro bloco, invalidando as demais cláusulas **catch**.

5. Funções para Manipulação de Classes

Para auxiliar os desenvolvedores em algumas situações em que é necessário verificar se uma determinada classe existe, ou se um método foi implementado, ou se um objeto é de uma determinada classe, existem no PHP várias funções para manipulação de classes.

```
bool class_exists(nome_classe, autoload)
```

Retorna TRUE se a classe informada existir.

```
string get_class(objeto)
```

Retorna o nome da classe ao qual o objeto pertence.

```
array get_declared_classes()
```

Retorna um array com todas as classes declaradas no script atual, incluindo as classes predefinidas pelo PHP.

```
array get_declared_interfaces()
```

Retorna um array com as interfaces declaradas no script atual.

```
bool objeto instanceof classe
```

Retorna true se o objeto informado for uma instância de uma determinada classe.

```
bool method_exists(objeto, método)
```

Retorna True se o método informado existe no objeto informado.

```
bool is_callable???????????????
```

Uma outra forma para determinar se um método está presente em determinada classe, sem ter de instanciá-la é através da função `is_callable`, que pode ser executada da seguinte forma:

```
if(is_callable(array("nome_classe", "método")))
```

6. Introdução ao PEAR

PEAR é um acrônimo para “PHP Extension and Application Repository”, isto é, Repositório de Aplicações e Extensões PHP. O propósito do PEAR é fornecer:

- Uma biblioteca estruturada de códigos abertos para programadores PHP.
- Um sistema para distribuição de código e manutenção de pacotes.
- Um padrão para escrita de código em PHP.
- A PECL, “PHP Extension Community Library”.
- Um site, uma lista de e-mails e mirrors para download.

PEAR é um projeto comunitário, governado por seus desenvolvedores.

Os códigos do PEAR são divididos em pacotes. Cada pacote é um projeto separado, com seu próprio time de desenvolvimento, número de versão, ciclo de liberação, documentação, etc. Pacotes podem estar relacionados com outros através de dependências explícitas, mas não há dependências automáticas baseadas nos nomes dos pacotes. Por exemplo, “HTTP_Post” por padrão não depende de “HTTP”. Porém isso não significa que esse tipo de dependência nunca irá ocorrer. Por exemplo: o pacote “DB_DataObject” depende do pacote “DB”. Para visualizar os pacotes disponíveis, acesso o site: <http://pear.php.net/packages.php>

Um guia para o estilo de padrões de codificação do PEAR podem ser encontrados em: <http://pear.php.net/manual/en/standards.php>

Todos os pacotes do PEAR são registrados e estão disponíveis para download em um servidor: <http://pear.php.net>. Servidores de terceiros, chamados de “channels” (canais, veja mais em: <http://pear.php.net/channels>) também podem distribuir pacotes que podem ser instalados pelo instalador do PEAR. O PEAR não distribui esses pacotes de terceiros e somente presta suporte aos pacotes distribuídos em pear.php.net. Os pacotes são distribuídos compactados através do gzip contendo um arquivo XML de descrição embutido. Este XML possui informações sobre o pacote, os arquivos que fazem parte dele, suas dependências, etc.

PECL (pronuncia-se “pickle”), é um projeto separado que distribui extensões do PHP (códigos em C compilados, como a extensão “PDO”). As extensões do PECL são

distribuídas em pacotes que podem ser instaladas com o instalador do PEAR através do comando `pecl`. Mais informações sobre o PECL em: <http://pecl.php.net>.

6.1 Instalação

Provavelmente sua distribuição deve possuir um pacote já compilado da última versão estável do PEAR ou próximo disso. No Ubuntu Linux, este pacote chama-se “php-pear”, utilizando o gerenciador de pacotes `apt` o pacote e suas dependências podem ser instalados automaticamente através do comando:

```
apt-get install php-pear
```

Obs: para utilizar o pear, o php instalado deve possuir suporte a xml, deve também estar instalado o pacote `php-cli`.

6.2 Usando o PEAR

O comando `pear` é a principal ferramenta de instalação do PEAR, ele contém vários sub-comandos, como `install` e `upgrade` que podem ser executados em qualquer plataforma suportada pelo PEAR.

O primeiro comando que você deve conhecer é o `help`, o `pear help` exibe uma lista de sub-comandos, caso algum sub-comando seja informado, como `pear help upgrade`, será exibido um texto de ajuda com todas as opções de linha de comando para tal sub-comando.

```
root@pavilion:~# pear help
Commands:
build          Build an Extension From C Source
bundle         Unpacks a Pecl Package
channel-add    Add a Channel
channel-alias  Specify an alias to a channel name
channel-delete Remove a Channel From the List
channel-discover Initialize a Channel from its server
channel-info   Retrieve Information on a Channel
channel-update Update an Existing Channel
clear-cache    Clear Web Services Cache
config-create  Create a Default configuration file
config-get     Show One Setting
config-help    Show Information About Setting
config-set     Change Setting
config-show    Show All Settings
convert        Convert a package.xml 1.0 to package.xml 2.0
format
cvsdiff        Run a "cvs diff" for all files in a package
cvstag         Set CVS Release Tag
```


download	Download Package
download-all	Downloads each available package from the
default channel	
info	Display information about a package
install	Install Package
list	List Installed Packages In The Default Channel
list-all	List All Packages
list-channels	List Available Channels
list-files	List Files In Installed Package
list-upgrades	List Available Upgrades
login	Connects and authenticates to remote server
logout	Logs out from the remote server
makerpm	Builds an RPM spec file from a PEAR package
package	Build Package
package-dependencies	Show package dependencies
package-validate	Validate Package Consistency
pickle	Build PECL Package
remote-info	Information About Remote Packages
remote-list	List Remote Packages
run-scripts	Run Post-Install Scripts bundled with a package
run-tests	Run Regression Tests
search	Search remote package database
shell-test	Shell Script Test
sign	Sign a package distribution file
uninstall	Un-install Package
update-channels	Update the Channel List
upgrade	Upgrade Package
upgrade-all	Upgrade All Packages
Usage: pear [options] command [command-options] <parameters>	
Type "pear help options" to list all options.	
Type "pear help shortcuts" to list all command shortcuts.	
Type "pear help <command>" to get the help for the specified command.	

Existem opções que podem ser passadas diretamente para o comando `pear` ou para algum sub-comando, obedecendo a seguinte sintaxe:

```
pear [options] sub-command [sub-command options] [sub-command arguments]
```

Para listar o conjunto de opções disponíveis para o comando `pear` em si, podemos utilizar o comando `pear help options`:

```
root@pavilion:~# pear help options
Options:
  -v          increase verbosity level (default 1)
  -q          be quiet, decrease verbosity level
  -c file     find user configuration in `file'
  -C file     find system configuration in `file'
  -d foo=bar  set user config variable `foo' to `bar'
  -D foo=bar  set system config variable `foo' to `bar'
  -G          start in graphical (Gtk) mode
  -s          store user configuration
```

-S	store system configuration
-u foo	unset `foo` in the user configuration
-h, -?	display help/usage (this message)
-V	version information

6.3 Parâmetros de configuração

Os diferentes front-ends diferem apenas nas partes específicas à apresentação da interface do usuário. A parte principal de execução de cada comando é compartilhada entre todas as front-ends, bem como seus parâmetros de configuração. O PEAR possui muitos parâmetros de configuração, mas você só precisa se preocupar com alguns deles, os principais, que estão descritos logo abaixo:

Obs: com o comando `pear config-show`, você pode visualizar alguns dos parâmetros de configuração.

PEAR main directory (php_dir) – Diretório em que os arquivos de inclusão PHP são armazenados, assim como os arquivos de administração interna do PEAR para monitorar os pacotes instalados. O valor padrão é: `/usr/local/lib/php`

PEAR executables directory (bin_dir) – Diretório em que scripts executáveis e programas são instalados. Por exemplo, o comando `pear` é instalado aqui. O valor padrão é: `/usr/local/bin`

PEAR documentation directory (doc_dir) – Diretório em que os arquivos de documentação são instalados. Logo abaixo de `doc_dir` há um diretório com o mesmo nome do pacote contendo todos os arquivos de documentação instalados com o pacote. O valor padrão é `/usr/local/lib/php/docs`.

PHP extension directory (ext_dir) – diretório em que todas as extensões do PHP ficam, por exemplo, os módulos para interação com o banco de dados mysql, postgres, etc. O valor padrão é `/usr/local/lib/php/extensions/<BUILDSPEC>`, onde BUILDSPEC é a versão da compilação do php atualmente instalado, ex: 20020429

PEAR installer cache directory (cache_dir) – Diretório em que o `pear` pode armazenar dados de cache. Esses dados são utilizados para agilizar chamadas XML-RPC repetidas para o servidor central.

PEAR data directory (data_dir) – Diretório que armazena arquivos que não são códigos, testes de regressão, executáveis nem documentação. Os candidatos típicos são arquivos DTD, planilhas XSL, arquivos de modelo offline, etc.

Cache TimeToLive (cache_ttl) – número em segundos que as chamadas XML-RPC devem permanecer em cache, acelerando as operações remotas. “0” desativa o cache.

Preferred Package State (preferred_state) – Define a qualidade esperada para a versão de um pacote antes mesmo de vê-lo. Há cinco estados disponíveis: stable, beta, alpha, snapshot e devel. A qualidade mais alta é a stable, caso ela esteja definida para preferred_state, apenas pacotes classificados como stable serão visualizados ao se navegar pelo banco de dados de pacotes. Caso alpha seja definido, os pacotes stable, beta e alpha estarão disponíveis no banco de dados de pacotes.

Unix File Mask (unmask) – utilizado para determinar as permissões padrões para novos arquivos criados em sistemas Unix. O valor de unmask será subtraído da máscara.

Debug Log Level (verbose) – Nível padrão do registro de depuração: 1 para informativo, 2 para alguns detalhes sobre as operações do instalador e 3 para debug.

HTTP_Proxy_Server (http_proxy) – Definindo este parâmetro o *pear* sempre utilizará um proxy para suas operações remotas. Ex: *http://user:pw@host:port* ou somente *host:port*

PEAR Server (master_server) – O nome do host do servidor de registro do pacote. As pesquisas de registro e os downloads são todos colocados em proxy por meio desse servidor.

Para alterar qualquer das configurações podemos utilizar o comando *pear-config-set*. Por exemplo:

```
pear config-set php_dir /usr/share/pear
                preferred_state alpha
```

6.4 Comandos PEAR

Nesta seção iremos aprender sobre os principais comandos do *PEAR Installer* para a instalação e manutenção de pacotes em seu sistema.

6.4.1 pear install

Instala um pacote pear copiando os arquivos contidos no pacote para seus respectivos diretórios no sistema alvo, de acordo com as instruções contidas no arquivo XML do pacote. O pacote pode ser um arquivo local, apenas o nome do

pacote para ser procurado nos canais cadastrados ou até mesmo uma URL completa. Consulte a saída do comando `pear help install` para detalhes.

Ex:

```
pear install Console_Table
pear install -o HTML_QuickForm2
pear -d preferred_state=beta install -a Services_weather
```

O primeiro comando simplesmente procura e instala o pacote `Console_Table` dos canais PEAR cadastrados. O segundo comando instala o pacote `HTML_QuickForm` apenas com as dependências requeridas, não instalando as opcionais. O terceiro comando procura nos pacotes de qualidade `beta` ou superior pelo pacote `Services_weather` instalando todas as dependências, tanto as requeridas quanto as opcionais.

O parâmetro `-d` está descrito na listagem do comando `pear help options`. Ele serve para definir algum parâmetro de configuração para o comando que estiver sendo executado, o comando `pear config-set` altera o parâmetro permanentemente.

6.4.2 pear list

O comando `pear list` lista os pacotes instalados ou o conteúdo do pacote informado na linha de comando.

```
root@pavilion:~# pear list
Installed packages, channel pear.php.net:
=====
Package      Version State
Archive_Tar   1.3.2   stable
Console_Getopt 1.2     stable
PEAR          1.4.11  stable

root@pavilion:~# pear list Archive_Tar
Installed Files For Archive_Tar
=====
Type Install Path
php  /usr/share/php/Archive/Tar.php
doc  /usr/share/php/doc/Archive_Tar/docs/Archive_Tar.txt
```

6.4.3 pear info

O comando `pear info` exibe informações sobre um pacote instalado, um pacote tarball ou um arquivo (XML) de definição de pacote.

```

root@pavilion:~# pear info Archive_Tar
About pear.php.net/Archive_Tar-1.3.2
=====
Release Type      PEAR-style PHP-based Package
Name              Archive_Tar
Channel           pear.php.net
Summary           Tar file management class
Description        This class provides handling of tar files in
                   PHP.
                   It supports creating, listing, extracting and
                   adding to tar files.
                   Gzip support is available if PHP has the zlib
                   extension built-in or
                   loaded. Bz2 compression is also supported with
                   the bz2 extension loaded.
Maintainers       Gregory Beaver <cellog@php.net> (lead)
                   Vincent Blavet <vincent@phpconcept.net> (lead)
                   Stig Bakken <stig@php.net> (helper)
Release Date      2007-01-05 22:35:26
Release Version    1.3.2 (stable)
API Version        1.3.2 (stable)
License            PHP License (http://www.php.net/license)
Release Notes      Correct Bug #4016
                   Remove duplicate remove error display with '@'
                   Correct Bug #3909 : Check existence of
                   OS_WINDOWS constant
                   Correct Bug #5452 fix for "lone zero block"
                   when untarring packages
                   Change filemode (from pear-core/Archive/Tar.php
                   v.1.21)
                   Correct Bug #6486 Can not extract symlinks
                   Correct Bug #6933 Archive_Tar (Tar file
                   management class) Directory traversal
                   Correct Bug #8114 Files added on-the-fly not
                   storing date
                   Correct Bug #9352 Bug on _dirCheck function
                   over nfs path
Compatible with    P/P
                   Versions >= P, <= Pp/p
                   Versions >= p, <= p1/1
                   Versions >= 1, <= 11/1
                   Versions >= 1, <= 1
Not Compatible with P/P
                   Versions P
                   p/p
                   Versions p
                   1/1
                   Versions 1
                   1/1
                   Versions 1
Required Dependencies PHP version 4.0.0

```

```

PEAR installer version 1.4.0b1 or newer
package.xml version      2.0
Last Modified            2007-05-22 15:05
Last Installed Version - None -

```

6.4.4 *pear list-all*

O comando `pear list-all` exibe uma lista classificada em ordem alfabética de todos os pacotes com a última versão estável e a versão que você tem instalada, se houver alguma. A saída deste comando é longa porque ele exibe todos os pacotes que possuem uma versão estável.

6.4.5 *pear list-upgrades*

Este comando compara a versão instalada com a versão mais recente disponível de acordo com o estado de versão configurado, `stable`, `beta`, `alpha`... (veja o parâmetro `preferred_state` para mais detalhes).

```

root@pavilion:~# pear list-upgrades
pear.php.net Available Upgrades (stable):
=====
Channel      Package      Local      Remote      Size
pear.php.net Console_Getopt 1.2 (stable) 1.2.3 (stable) 3.9kB
pear.php.net PEAR          1.4.11 (stable) 1.6.1 (stable) 289kB

```

6.4.6 *pear upgrade*

Atualiza o pacote informado para a última versão disponível, de acordo com o estado de versão configurado.

```

root@pavilion:~# pear upgrade PEAR
Did not download optional dependencies: pear/XML_RPC, use --alldeps to
download automatically
/tmp/glibctestnXyzby:1:22: error: features.h: Arquivo ou diretório
inexistente
pear/PEAR can optionally use package "pear/XML_RPC" (version >= 1.4.0)
downloading PEAR-1.6.1.tgz ...
Starting to download PEAR-1.6.1.tgz (295,780 bytes)
.....done: 295,780 bytes
downloading Structures_Graph-1.0.2.tgz ...
Starting to download Structures_Graph-1.0.2.tgz (30,947 bytes)
...done: 30,947 bytes
downloading Console_Getopt-1.2.3.tgz ...
Starting to download Console_Getopt-1.2.3.tgz (4,011 bytes)
...done: 4,011 bytes

```

```
upgrade ok: channel://pear.php.net/Console_Getopt-1.2.3
upgrade ok: channel://pear.php.net/Structures_Graph-1.0.2
upgrade ok: channel://pear.php.net/PEAR-1.6.1
PEAR: Optional feature webinstaller available (PEAR's web-based
installer)
PEAR: Optional feature gtkinstaller available (PEAR's PHP-GTK-based
installer)
PEAR: Optional feature gtk2installer available (PEAR's PHP-GTK2-based
installer)
To install use "pear install pear/PEAR#featurename"
```

6.4.7 *pear upgrade-all*

Este comando é uma combinação dos comandos `list-upgrade` e `upgrade`, o qual irá atualizar todo pacote que possuir uma versão mais nova disponível para instalação.

6.4.8 *pear uninstall*

O comando `pear uninstall` é utilizado quando desejamos remover um pacote do sistema local.

6.4.9 *pear search*

O comando `pear search` efetua uma busca por substrings de nomes de pacotes.

6.4.10 *pear remote-list*

Exibe uma lista de todos os pacotes e versões estáveis disponíveis no repositório de pacotes. Podemos alterar temporariamente o parâmetro `preferred_state` para buscar pacotes com versões beta, alpha, etc.

```
pear -d preferred_state=alpha remote-list
```

6.4.11 *pear remote-info*

Para exibir informações detalhadas sobre um pacote que não está instalado, use o comando `pear remote-info`.

6.4.12 *pear download*

O comando `pear install` não armazena o arquivo do pacote do qual a instalação é efetuada em lugar algum. Se você quer apenas o pacote tarball (para instalar depois, ou por qualquer outro motivo), pode usar o comando `pear download`.

6.4.13 *pear config-get*

Usado para exibir um parâmetro de configuração.

```
pear config-get preferred_state
```

6.4.14 *pear config-set*

Usado para definir um parâmetro de configuração

```
pear config-set preferred_state=alpha
```

6.4.15 *pear config-show*

Exibe todos os parâmetros de configuração

6.4.16 *pear shortcuts*

Exibe atalhos para os comandos.

7. Abstração de Banco de Dados com PEAR-MDB2

A maioria dos programadores php já utilizou ou pelo menos já ouviu falar do pacote pear-DB para abstrair interações com diversos tipos de bancos de dados. Até 2006 esse pacote era considerado o pacote oficial do php para abstração de bancos de dados, porém em meados de 2007 um novo pacote o pear-mdb2 tomou este posto. Devido a uma maior portabilidade e um maior número de de bancos de dados suportados, além de inúmeras funcionalidades adicionais o pacote php-mdb2 é agora recomendado pelos desenvolvedores do PEAR e o uso do pacote php-db será descontinuado.

O pacote PEAR-MDB2 é uma junção do PEAR-DB e do Metabase, camadas para abstração de databases. Isso provê uma API em comum para os diversos tipos de bancos de dados suportados pelo MDB2. A principal diferença dos outros pacotes para abstração de bancos de dados é que o MDB2 vai muito mais além para garantir uma maior portabilidade. Mais características sobre o pear-mdb2 em: <http://pear.php.net/manual/en/package.database.mdb2.intro.php>

7.1 Instalação

Ao instalar o pacote pear-MDB2 você terá apenas as classes básicas, porém para ter total interação com determinado tipo de banco de dados você terá de instalar o driver adequado ao banco de dados que você deseja trabalhar. Os comandos abaixo instalam o pacote básico pear-MDB2 e os drivers para mysql e postgresSQL:

```
pear install MDB2
pear install MDB2#mysql
pear install MDB2#pgsql
```

Obs: o pacote MDB2 funciona apenas como uma camada de abstração, portanto, você deve possuir instalados, os módulos do php para interação com o banco de dados desejado, por exemplo, php5-mysql.

7.2 DSN

DSN ou Data Source Name, é uma string com um formato pré-estabelecido que é utilizada pelas classes básicas do pear-MDB2 e por quase todos os pacotes pear para abstração de banco de dados, com o objetivo de fornecer as informações necessárias ao se conectar ao banco de dados desejado.

Um DSN consiste nas seguintes partes:

`phptype`: Database Backend que será utilizado (ex: mysql, pgsql, etc)

`dbsyntax`: Database utilizada com respeito a sintaxe SQL

`protocol`: Protocolo de comunicação (ex: tcp, unix, etc)

`hostspec`: Especificação do host (hostname:[port])

`database`: Database que será utilizada no servidor

`username`: usuário para login

`password`: senha para login

`proto_opts`: pode ser utilizado em conjunto com o protocolo de comunicação

`options`: opções de conexão adicionais em formato de string URI, opções são separadas pelo símbolo '&'

O DSN pode ser um array associativo ou uma string no seguinte formato:

```
phptype(dbsyntax)://username:password@protocol+hostspec/database?option=value
```

Os seguintes backends para bancos de dados são atualmente suportados:

`fbsql` -> FrontBase

`ibase` -> InterBase / Firebird (requires PHP 5)

`mssql` -> Microsoft SQL Server

`mysql` -> MySQL

`mysqli` -> MySQL (supports new authentication protocol)
(requires PHP 5)

`oci8` -> Oracle 7/8/9/10

`pgsql` -> PostgreSQL

```
querysim -> QuerySim  
sqlite -> SQLite 2
```

7.3 Conectando

Para instanciar um objeto do tipo database com o MDB2 existem três maneiras:

factory() : será instanciada uma nova instância da classe `MDB2_Driver_common`, porém não será feita uma conexão ao banco de dados até que ela seja necessária. Esta forma é considerada mais eficiente, já que pode-se ter caches realizados pela aplicação, e assim nem sempre será necessário efetuar uma conexão com o banco de dados.

connect(): será instanciada uma nova instância da `MDB2_Driver_common`, e será estabelecida uma conexão com o banco de dados imediatamente. Desta forma qualquer problema na conexão com o banco irá gerar um erro imediatamente.

singleton(): retorna uma instância da `MDB2_Driver_common`. Um novo objeto do tipo `MDB2_Driver_common` será criado uma única vez através do método `factory()` descrito acima, chamadas subsequentes do tipo `singleton` irão retornar uma referência para o objeto já instanciado. Esta forma é recomendada ao invés de você declarar seu objeto database como global.

Para se conectar a um banco de dados devemos utilizar uma das seguintes funções acima, `factory()`, `connect()` ou `singleton()`, as quais requerem um DSN válido como primeiro parâmetro. Este parâmetro pode ser uma string ou um array. A string padrão de um DSN já foi descrita anteriormente, e será utilizada nos exemplos posteriores, porém caso deseje utilizar na forma de um array, o mesmo deve possuir a seguinte estrutura:

```
$dsn = array(  
    'phptype' => false,  
    'dbsyntax' => false,  
    'username' => false,  
    'password' => false,  
    'protocol' => false,  
    'hostspec' => false,  
    'port' => false,  
    'socket' => false,  
    'database' => false,  
    'new_link' => false,  
);
```

O segundo parâmetro é um array opcional \$options, o qual contém um conjunto de opções deste pacote que podem ser modificadas em tempo de execução.

Name	Type	Description
ssl	boolean	determines if ssl should be used for connections
field_case	integer	CASE_LOWER CASE_UPPER: determines what case to force on field/table names
disable_query	boolean	determines if queries should be executed
result_class	string	class used for result sets
buffered_result_class	string	class used for buffered result sets, default is MDB2_Result_Common
result_wrap_class	string	class used to wrap result sets into, default is MDB2_Result_Common
result_buffering	boolean	should results be buffered or not?
fetch_class	string	class to use when fetch mode object is used
persistent	boolean	persistent connection?
debug	integer	numeric debug level
debug_handler	string	function/method that captures debug messages
debug_expanded_output	boolean	BC option to determine if more context information should be send to the debug handler
default_text_field_length	integer	default text field length to use
lob_buffer_length	integer	LOB buffer length
log_line_break	string	line-break format
idxname_format	string	pattern with '%s' for index name
seqname_format	string	pattern with '%s' for sequence name
savepoint_format	string	pattern with '%s' for auto generated savepoint names

Name	Type	Description
seqcol_name	string	sequence column name
quote_identifier	boolean	if identifier quoting should be done when check_option is used
use_transactions	boolean	if transaction use should be enabled
decimal_places	integer	number of decimal places to handle
portability	integer	portability constant
modules	array	short to long module name mapping for __call()
emulate_prepared	boolean	force prepared statements to be emulated
datatype_map	array	map user defined datatypes to other primitive datatypes
'datatype_map_callback	array	callback function/method that should be called

Em caso de sucesso, teremos uma nova instância da classe database. É extremamente recomendado que você cheque o valor de retorno com PEAR::isError(), ou algum método isError() específico do pacote MDB2. Para desconectar utilizamos o método disconnect() da instância da database.

Abaixo temos um exemplo prático de como conectar em um banco usando um DSN:

```
<?php
require_once 'MDB2.php';

$dsn = array(
    'phptype' => 'mysql',
    'username' => 'aluno',
    'password' => 'milenium',
    'hostspec' => 'localhost',
    'database' => 'curso',
);

$options = array(
    'debug' => 2,
    'portability' => MDB2_PORTABILITY_ALL,
);

// uses MDB2::factory() to create the instance
// and also attempts to connect to the host
```

```
$link = & MDB2::connect($dsn, $options);  
if (PEAR::isError($link)) {  
    die($link->getMessage());  
}  
?>
```

O pacote PEAR MDB2 provê vários métodos para execução de queries em bancos de dados. O método mais direto é o `query()`, o qual recebe uma string como argumento, e possui dois retornos possíveis: um novo objeto `MDB2_Result` para queries que retornam resultados (como queries do tipo `SELECT`), ou um objeto `MDB2_Error` em caso de falha.

O método `query()` não deve ser utilizado para queries que manipulam dados (`INSERT` por exemplo), em seu lugar devemos utilizar o método `exec()`, o qual também terá dois possíveis retornos: um inteiro representando o número de linhas afetadas pela query, ou um objeto `MDB2_Error` em caso de falha.

7.3.1 Limitando linhas ou lendo a partir de um offset

Para ler ou gravar um número limitado de linhas através de um query, ou para iniciar uma leitura a partir de determinado offset podemos utilizar o método `setLimit()` antes da execução de uma query. O limite e o offset irão afetar somente a próxima chamada de execução de uma query, a qual sempre irá limpar qualquer valor definido pelo método `setLimit()`.

Abaixo temos um exemplo de como executar uma query, definindo um limite e/ou um offset para a mesma.

```
<?php  
// Once you have a valid MDB2 object named $link...  
  
$sql = "SELECT * FROM clients";  
// read 20 rows with an offset of 10  
$link->setLimit(20, 10);  
$affected =& $link->exec($sql);  
  
$sql = "DELETE FROM clients";  
if ($link->supports('limit_queries') === 'emulated') {  
    echo 'offset will likely be ignored'  
}  
// only delete 10 rows  
$link->setLimit(10);  
$affected =& $link->exec($sql);  
?>
```

7.4 Resultados

Obtendo dados do resultado de uma query.

O objeto `MDB2_Result_Common` dispõe de quatro métodos para recuperar os dados das linhas de um `ResultSet`: `fetchOne()`, `fetchRow()`, `fetchCol()` e `fetchAll()`.

`fetchRow()` e `fetchOne()` lêem toda uma linha ou todos os campos de uma coluna respectivamente. O ponteiro do resultado é movido para a próxima linha toda vez que esses métodos são chamados, retornando `Null` quando não existem mais linhas a serem recuperadas.

`fetchAll()` e `fetchCol()`, lêem todas as linhas de uma só vez de um `ResultSet`, movendo o ponteiro do resultado para o final. Enquanto o método `fetchAll()` lê todos os dados de uma linha, o método `fetchCol()` lê apenas todas as linhas de uma coluna.

```
<?php
// Create a valid MDB2 object named $mdb2
// at the beginning of your program...
require_once 'MDB2.php';

$link =& MDB2::connect('mysql://usr:pw@localhost/dbnam');
if (PEAR::isError($link)) {
    die($link->getMessage());
}

// Proceed with getting some data...
$res =& $link->query('SELECT * FROM mytable');

// Get each row of data on each iteration until
// there are no more rows
while (($row = $res->fetchRow())) {
    echo $row[0] . "\n";
}

// while (($one = $res->fetchOne())) {
//     echo $one . "\n";
// }

?>
```

7.4.1 Formato das linhas recuperadas

Os dados de uma linha de um resultado de uma query podem ser disponibilizados em um dos três tipos seguintes:

- `MDB2_FETCHMODE_ORDERED`

Array Ordenado, onde as chaves são os números das colunas.

- MDB2_FETCHMODE_ASSOC

Array Associativo, onde as chaves são os nomes das colunas.

- MDB2_FETCHMODE_OBJECT

Objeto, com os nomes das colunas como propriedades.

Para alterar o formato das linhas recuperadas, que por padrão é um Array Ordenado, podemos utilizar duas formas, por chamada, passando uma das constantes acima para um dos quatro métodos de recuperação de linhas de um resultado, ou alterar o padrão para todas as chamadas através do método `setFetchMode()`, passando uma das constantes acima como parâmetro.

```
<?php
// Once you have a valid MDB2 object named $mdb2...
$res =& $link->query('SELECT * FROM users');

while ($row = $res->fetchRow(MDB2_FETCHMODE_ASSOC)) {
    echo $row['id'] . "\n";
}
?>

<?php
// Once you have a valid MDB2 object named $mdb2...
$link->setFetchMode(MDB2_FETCHMODE_ASSOC);

$res =& $link->query('SELECT * FROM users');

while ($row = $res->fetchRow()) {
    echo $row['id'] . "\n";
}
?>
```

Para mais detalhes:

<http://pear.php.net/manual/en/package.database.mdb2.intro-fetch.php>

8. Scripts Shell PHP

A versão CLI (Command Line Interface) o PHP serve para escrever scripts de shell autônomos executados independentemente de qualquer servidor Web. A partir da versão 4.3, a versão CLI do PHP é instalada por padrão, e apesar de ainda possuir uma série de soluções alternativas, CGI por exemplo, a CLI é a única alternativa realmente eficiente quando queremos utilizar PHP sem um servidor Web. A CLI reduz o PHP ao básico, não importando variáveis de formulário GET ou POST, não produz cabeçalhos MIME na saída e geralmente não tem outras atividades que outras implementação SAPI têm. A versão CLI de PHP se comporta como qualquer outro analisador de script, como Perl ou Python, a única prova restante da herança da Web de PHP é o fato de que você ainda precisa usar as tags `<?php ?>` para delimitar seu código.

Nota: a CLI possui valores padrões diferentes para o arquivo `php.ini`, na maioria das distribuições temos arquivos `php.ini` separados para o servidor Web e para a CLI.

8.1 Entrada do usuário

Se você precisa entrar com dados em um script shell PHP, use a entrada-padrão, que está disponível no fluxo PHP STDIN ou no dispositivo “terminal typewriter” (`/dev/tty`).

```
<?php
print "Qual o seu nome? ";
$nome = trim(fgets(STDIN));
?>
```

8.2 Analisando opções de linha de comando

Não existe uma função `getopt` interna no PHP, mas o PEAR oferece um pacote chamado `Console_Getopt` que suporta tanto opções curtas como longas (estilo GNU). `Console_Getopt` é empacotado com o PHP e é instalado como padrão a menos que você desative explicitamente o PEAR. Abaixo temos um script de linha de comando aceitando quatro opções curtas: `-v` e `-q`, para aumentar e diminuir as mensagens geradas pelo próprio script, `-h` para exibir ajuda e `-c` para definir outro arquivo de configuração.

```
#!/usr/bin/php

<?php
require_once("Console/Getopt.php");
$verbose = 1;
$config_file = $_ENV['HOME'] . '/.myrc';
$options = Console_Getopt::getopt($argv, 'hqvc:');
foreach( $options[0] as $opt) {
    switch($opt[0]) {
        case 'q':
            $verbose--;
            break;
        case 'v':
            $verbose++;
            break;
        case 'h':
            usage();
            break;
        case 'c':
            $config_file = $opt[1];
            break;
    }
}
if( $verbose > 1 ) {
    print "Config file is \"$config_file\".\n";
}

function usage() {
    $stderr = fopen("php://stderr", "w");
    $progname = basename($GLOBALS['argv'][0]);
    fwrite($stderr, "Usage: $progname [-qvh] -c [config-file]
Options:

-q          menos detalhado
-v          mais detalhado
-h          ver ajuda
-c <file>   ler configuracoes de <file>");
    fclose($stderr);
}

echo "Sua linha de comando foi: " . implode(" ", $argv);

?>
```

Primeiro, o script inclui a definição de classe `Console_Getopt`. Depois de definir valores-padrão para `$verbose` e `$config_file`, a chamada `getopt()` é feita com a lista de parâmetros e uma string especificando quais opções são aceitas.

Cada caractere alfanumérico na string de especificação de opções é uma opção válida. Se o caractere da opção for seguido por dois pontos, a opção terá um valor. No exemplo anterior, `c:` diz que a opção `-c` espera um parâmetro, que é o

arquivo de configuração a ser usado. As opções -q, -v e -h não são seguidas de caracteres especiais, são opções simples do tipo sinalização/alterância.

O método getopt() retorna uma matriz da forma array (array (option, value), ...). O loop foreach itera por meio dessa matriz e \$opt é atribuído a array(option, value). Para opções de sinalização, o valor sempre será NULL (não é preciso verificar porque você já sabe quais opções são sinalizações simples), enquanto que para opções que pegam parâmetros, o segundo elemento nessa matriz é o parâmetro real. Por exemplo, -c foo daria array(ç, 'foo) em \$foo. É possível tratar a mesma opção quantas vezes for necessário. Nesse exemplo, o nível de verbosidade do programa aumenta em 1 toda vez que a opção -v é usada. Se o usuário especificar -vvvvv, o nível de verbosidade será aumentado em 5 vezes.

Também é possível especificar que um parâmetro de opção é opcional usando duas vezes dois pontos em vez de uma – por exemplo, c::: Ao encontrar um parâmetro de opção que não é obrigatório, Console_Getopt usa os restos da opção como o valor do parâmetro da opção. Por exemplo, se a opção -c foi especificada com c::, a string da opção -cfoo.cf daria o valor de parâmetro da opção foo.cf, mas apenas -c seria permitido também. No entanto, quando um parâmetro de opção torna-se opcional, -c foo não é mais permitido, tem que ser -cfoo.

8.3 Práticas Indicadas

Ao escrever scripts shell, siga algumas práticas indicadas para tornar a vida mais fácil para você e para os outros que irão usar seu script.

Por exemplo, a maioria dos servidores UNIX espera que seus programas respondam a foo -h ou foo --help com uma breve mensagem de uso ou que imprimam erros em erro-padrão. Nesta seção iremos aprender algumas práticas consideradas boas práticas.

8.3.1 Mensagem de uso

Depois de usar o UNIX/Linux por algum tempo, você se acostuma a poder digitar command --help ou command -h para obter uma breve descrição da opção de um comando e uso geral. A maioria dos usuários UNIX espera que seu programa responda a essas opções. Exiba uma mensagem de uso em erro-padrão e saia com um código diferente de 0 se o script for iniciado sem os parâmetros esperados ou se for executado com a opção -h (--help se você estiver usando opções longas). A

mensagem de uso deve listar todos os parâmetros obrigatórios e opcionais e fica assim:

```
Use: meuscript [opções] <file...>
```

Opções:

```
-v, --version      mostra a versão do meuscript
-h, --help         indica este texto de ajuda
-d dsn, --dsn=dsn  conecta ao banco de dados "dsn"
```

Também existe uma notação padrão para as opções e os parâmetros:

<code>[-c]</code>	Pode ter -c
<code>{-c foo}</code>	Pode ter -c com um parâmetro
<code>[-abcdef]</code>	Pode ter de -a a -f
<code>[-a -b]</code>	Pode ter -a ou -b
<code>{-a -b}</code>	Deve ter -a ou -b
<code><file></code>	Deve ter file como um parâmetro (não opção)
<code><file...></code>	Deve ter um ou mais parâmetros file
<code>[file...]</code>	Pode ter um ou mais parâmetros file

Se seu programa aceitar apenas algumas opções, liste-as na primeira linha da mensagem de uso, assim:

```
Uso: meuscript [-vh] [-d dsn] <file...>
```

Opções:

```
-v, --version      mostra a versão do meuscript
-h, --help         indica este texto de ajuda
-d dsn, --dsn=dsn  conecta ao banco de dados "dsn"
```

8.3.2 Código de Saída

Se o script falhar, saia com um código diferente de 0 (exceto 255, que é reservado pelo PHP para compilar/analisar erros). Se o script não falhar, saia com código 0.

8.3.3 Mensagens de Erro

Adicione o nome do script a todas as mensagens de erro, para que o usuário possa ver de qual script o erro se originou. Isso é útil se o script é invocado de outros scripts ou programas, permitindo ver de qual programa o erro veio.

9. Smarty

Como eu faço meus scripts em PHP independentes do layout?

Esta é sem dúvida uma das perguntas mais freqüentes nas listas de discussões sobre PHP: O PHP é vendido como sendo uma "linguagem de script embutida no HTML", após escrever alguns projetos que misturam HTML e PHP naturalmente vem uma idéia de que a separação da forma e conteúdo é uma boa prática. Além disso, em muitas empresas os papéis de designer e programador são separados.

9.1 O que é o Smarty?

O Smarty é um sistema de templates para PHP. Mais especificamente, ele fornece uma maneira fácil de controlar a separação entre a lógica da aplicação e a lógica da apresentação. Isto é melhor descrito em uma situação onde o programador da aplicação e o designer do template executam diferentes funções, ou na maioria dos casos não são a mesma pessoa.

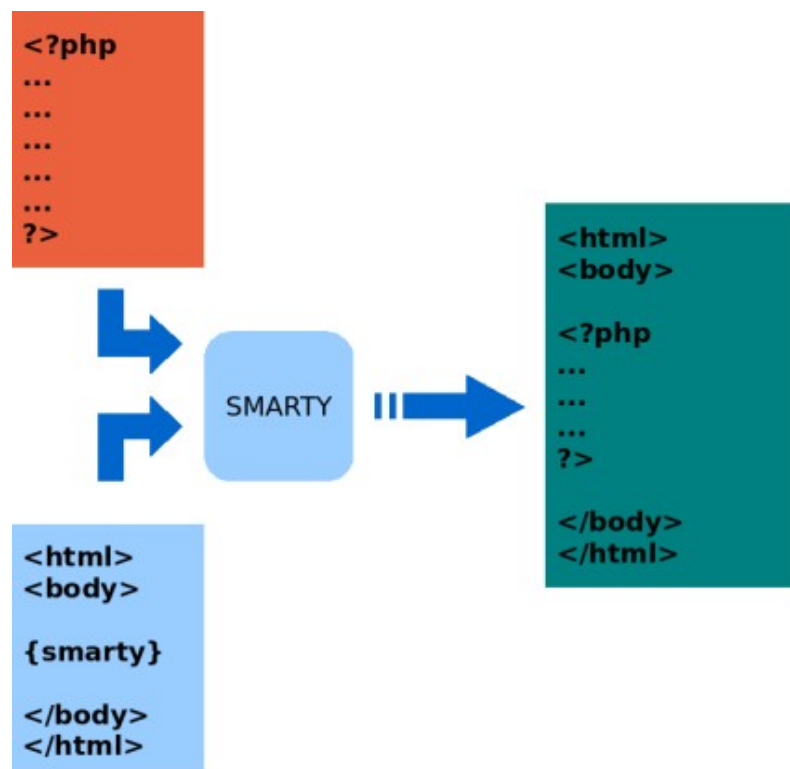
Um dos aspectos únicos do Smarty é seu sistema de compilação de templates. O Smarty lê os arquivos de templates e cria scripts PHP à partir deles. Uma vez criados, eles são executados sem ser necessário uma outra compilação do template novamente. Com isso, os arquivos de template não são 'parseados'(analisados) toda vez que um template é solicitado, e cada template tem a total vantagem de soluções de cache do compilador PHP, tais como: Zend Accelerator (<http://www.zend.com/>) ou PHP Accelerator (<http://www.php-accelerator.co.uk>).

Algumas das características do Smarty:

- Ele é extremamente rápido.
- Ele é eficiente, visto que o interpretador do PHP faz o trabalho mais pesado.
- Sem elevadas interpretações de template, compila apenas uma vez.
- Ele está atento para só recompilar os arquivos de template que foram mudados.
- Você pode fazer funções próprias e seus próprios modificadores de variáveis, assim a linguagem de templates é extremamente extensível.

- Delimitadores de tag configuráveis, sendo assim você pode usar {}, {}, <!--{}-->, etc.
- Os construtores if/elseif/else/endif são passados para o interpretador de PHP, assim a sintaxe de expressão {if ...} pode ser tanto simples quanto complexa da forma que você queira.
- Aninhamento ilimitado de sections, ifs, etc. permitidos.
- É possível embutir o código PHP diretamente em seus arquivos de template, apesar de que isto pode não ser necessário (não recomendado) visto que a ferramenta é tão customizável.
- Suporte de caching embutido.
- Fontes de template arbitrários.
- Funções de manipulação de cache customizadas.
- Arquitetura de plugin.

Utilizando o smarty, teremos dois arquivos, sendo um com a programação PHP e outro com o layout (HTML) e os elementos da Smarty (variáveis, funções etc). Utilizando esses dois arquivos, a Smarty irá gerar o arquivo final, que mistura HTML e PHP.



Como programador, você será responsável pela página *teste.php*. O designer será responsável pela página *teste.tpl*. A smarty será responsável por avaliar as duas páginas e gerar o script final.

Muitos desenvolvedores Web reclamam da perda de desempenho de suas aplicações devido ao uso de ferramentas de templates. A smarty tenta reduzir essa perda pelo processo de reutilizar os scripts já compilados. Por essa razão, a diferença de velocidade muitas vezes nem é percebida, principalmente quando as páginas envolvidas não são muito complexas.

9.2 Instalação

Para executar o smarty é preciso fazer o download do mesmo no site <http://smarty.php.net> ou procurar um pacote pré-compilado para sua distribuição.

Neste site está disponível para download um arquivo compactado, contendo todos os arquivos da ferramenta. Fazendo a descompactação, você verá que os arquivos da classe estão em um subdiretório chamado *libs*. Coloque esses arquivos no mesmo diretório das *libs* do php do seu servidor, para serem automaticamente incluídos através das chamadas *include* ou *require*.

9.3 Testando

Caso o exemplo abaixo produza algum erro, o php não está conseguindo encontrar o arquivo de definição de classe do smarty:

```
<?php
require_once('Smarty.class.php');
$smarty = new Smarty();
?>
```

Se nenhum erro for exibido, podemos prosseguir. Caso contrário precisamos definir a constante *SMARTY_DIR* antes de incluir a chamada para a classe. Por exemplo:

```
<?php
define('SMARTY_DIR', '/usr/share/php/smarty/libs/');
require_once(SMARTY_DIR . 'Smarty.class.php');
$smarty = new Smarty();
?>
```


9.4 Estrutura da aplicação

Para utilizar uma aplicação Smarty é necessária a criação de alguns diretórios. A Smarty requer quatro diretórios que, por padrão, são nomeados como:

templates	Armazena os arquivos dos templates (modelos).
templates_c	Diretório que armazena os templates compilados. Deve possuir permissão de escrita.
configs	Armazena arquivos de configuração, se houverem.
cache	Armazena os arquivos de cache, e não precisa ser criado caso essa funcionalidade esteja desabilitada. Deve possuir permissão de escrita.

Para exemplificar, faremos uma pequena aplicação, utilizaremos dois arquivos: `index.tpl`, que deve estar no diretório `templates` e `index.php`, que deve estar no diretório raiz da sua aplicação. Por exemplo:

```
./
|-- cache
|-- config s
|-- index.php
|-- templates
|   |-- index.tpl
|   |-- templates_c
```

`index.tpl`

```
{* Smarty *}
Olá {$nome}!
```

A primeira linha é apenas um comentário. Na segunda linha `{$nome}` é um trecho que será substituído pelo conteúdo da variável `$nome` do arquivo `index.php`.

`index.php`

```
<?php
require('Smarty.class.php');
$smarty = new Smarty;
$smarty->assign('nome','Rodrigo');
$smarty->display('index.tpl');
?>
```

Antes de tudo temos que incluir o arquivo que possui a definição da classe Smarty. Após isso temos que instanciar a classe Smarty. Na terceira instrução fazemos uma chamada ao método `assign`, que armazena a string 'Rodrigo' na variável de template `$nome`. Por último o método `display`, para compilar a template e o arquivo php, gerando o script que será exibido para o usuário.

Notas:

- O diretório `templates_c` deve ter permissão de escrita para o usuário do apache.
- O arquivo gerado em `templates_c` será exibido sempre que houver uma requisição para o arquivo `index.php`, até que o mesmo seja alterado.

9.5 Informações sobre o layout

Tudo o que será abordado neste tópico se aplica aos arquivos de templates (.tpl) que iremos criar.

9.5.1 Delimitadores

Os elementos da Smarty devem estar sempre entre os delimitadores utilizados pela ferramenta “{” e “}”.

9.5.2 Comentários

Podemos inserir comentários dentro dos delimitadores utilizando “*” após o delimitador de abertura e antes do delimitador de fechamento. Ex: {* comentário *}

9.5.3 Funções

Podemos também incluir chamadas para determinada função, que podem ser de dois tipos:

- **Embutidas** – são as funções internas da Smarty, e não podem ser modificadas. Ex: `if`, `elseif`, `else`, `section`.
- **Personalizadas (custom)** – são funções adicionais, acrescentadas através de plugins. Você pode modificar essas funções se desejar. Exemplo: `assign`, `eval`, `fetch`, `html_options`. Também estão neste grupo funções que você criar.

A sintaxe básica para a chamada de funções é a seguinte:

```
{nome_função atributo1="valor" atributo2="valor" ... }
```

Exemplos:

```
{include file="cabecalho.tpl"}
{if $nome eq "Kleber"}
    Seja bem vindo!
{else}
    Desculpe {$nome}, você não pode acessar esta página.
{/if}
```

A seguir um outro exemplo, envolvendo a função *section*, que é muito utilizada para percorrer valores de um array.

```
Nome dos clientes:
{section name=i loop=$nomes}
    {$nomes[i]}<br/>
{/section}
```

9.5.4 Atributos

Atributos na Smarty são bem semelhantes ao uso de atributos nas tags HTML. Servem para passar informações à função, definindo como será o seu funcionamento.

Caso esteja passando uma string como atributo, utilize aspas. Caso seja uma variável, as aspas são dispensáveis.

9.5.5 Variáveis

Existem três tipos de variáveis que podem ser usadas em uma aplicação Smarty:

- **Variáveis definidas pelo PHP** – iniciam com o caracter \$. Podem ser simples, arrays ou objetos. Ex: {\$nome}, {\$dados[0]}, {\$dados.nome}, {\$dados->nome}
- **Variáveis definidas em arquivos de configuração** – são definidas nos arquivos localizados no diretório configs da aplicação. Normalmente utiliza-se a extensão .conf para esses arquivos. Para utilizar essas variáveis no seu template, delimite-as com o caractere "#", ou então utilize a variável especial \$smarty.config. Por exemplo: {#nome#} ou {\$smarty.config.nome}
- **Variáveis reservada {\$smarty}** – através desta variável, podemos acessar diversas informações sobre o template, sobre o ambiente, obter o valor de cookies, de variáveis recebidas via GET ou POST etc. Ex:

```
{$_smarty.cookies.nomeusuario},  
{$_smarty.session.nome}.
```

```
{$_smarty.server.SERVER_NAME},
```

9.5.6 Modificadores de Variáveis

O objetivo dos modificadores é produzir alguma alteração em uma variável, função personalizada ou string. Eles devem ser acrescentados ao lado desses elementos, sendo precedidos pelo caracter | (*pipe*). Exemplos:

```
{* exibe o conteúdo da variável $nome em maiúsculas *}  
{ $nome | upper }  
  
{* exibe o conteúdo de $nome em minúsculas com espaços entre os  
caracteres *}  
{ $nome | lower | spacyfy }  
  
{* Exibe apenas até os 30 primeiros caracteres da variável $nome *}  
{ $nome | truncate:30 }
```

Uma lista completa dos modificadores de variáveis pode ser obtida em:
http://smarty.php.net/manual/pt_BR/language.modifiers.php .

9.5.7 Arquivos de Configuração

Os arquivos de configuração devem estar dentro do diretório “configs” da aplicação, são de grande utilidade para os designers. Pode-se manter nesses arquivos todas as variáveis globais, que serão usadas em várias páginas.

Para utilizar um arquivo de configuração em um *template*, devemos carregá-lo com a função `config_load`. Por exemplo: `{config_load file="teste.conf"}`.

Algumas considerações em relação aos arquivos de configuração:

- Comentários devem ter “#” no início da linha;
- Não é obrigatório o uso de aspas em torno dos valores das variáveis;
- Valores que ocupam mais de uma linha devem ser delimitados por aspas triplas (“”);
- É possível criar seções de variáveis, e posteriormente carregar apenas as seções desejadas. O nome das seções deve estar entre colchetes [].
- As variáveis globais, definidas fora de qualquer seção, sempre serão carregadas.

- Se existir uma variável global e uma de seção com o mesmo nome, será considerada a da seção.
- É possível ocultar variáveis ou seções inteiras utilizando “.” antes dos seus nomes. Isso é útil nos casos em que as variáveis interessam somente ao programador.

Exemplos:

```
{config_load file="index.conf"}  
  
{config_load file="index.conf" section="Margens"}
```

9.6 Informações sobre a programação PHP

Agora que já conhecemos os princípios básicos para a definição do layout através da criação de templates, iremos tratar da lógica do programa PHP, que inclui as variáveis e métodos da classe Smarty que podemos utilizar em nossos programas, antes de exibir o template.

9.6.1 A constante *SMARTY_DIR*

Conforme citado anteriormente, podemos definir a constante `SMARTY_DIR` para referenciar o diretório onde se encontra a classe do smarty.

```
define('SMARTY_DIR', '/usr/share/php/smarty/libs/');  
require_once(SMARTY_DIR . 'Smarty.class.php');
```

Se você utilizar a constante `SMARTY_DIR` antes de defini-la, a smarty tentará automaticamente definir seu valor.

9.6.2 Variáveis de configuração

A classe Smarty possui diversas variáveis de configuração que podem ser alteradas pelo programador. Você pode alterar o valor dessas variáveis de duas formas:

1. diretamente no arquivo da classe (`Smarty.class.php`). Nesse caso, o valor que você definir torna-se o padrão. Porém, esse método não é recomendável, pois se você instalar uma nova versão da classe Smarty irá perder os valores que definiu.
2. de dentro do programa PHP, digitando:

```
$smarty->nome_variavel = valor;
```

Onde `$smarty` é o objeto da classe Smarty, “nome_variavel” é um dos nomes apresentados no link a seguir e “valor” é o novo valor dessa variável para o objeto em questão.

Para ver a lista de variáveis que podem ser alteradas das duas maneiras acima, acesse: http://smarty.php.net/manual/pt_BR/api.variables.php.

9.6.3 Métodos

Além de poder manipular as variáveis da classe, temos à nossa disposição um conjunto de métodos que podem ser chamados por um objeto da classe Smarty em um programa PHP. Para chamar um método, procedemos da seguinte forma:

```
$smarty->nome_metodo([lista_parametros]);
```

Onde `$smarty` é o objeto da classe smarty, “nome_metodo” é o nome de um método existente na classe Smarty e [lista_parametros] são os parâmetros (se existirem) do método especificado. Os parâmetros são separados por vírgulas.

Um dos métodos mais utilizados é o `assign`, que tem a função de atribuir valores às variáveis que serão utilizadas nos templates. Por exemplo:

```
$smarty->assign("nome", "Kleber Silva");
```

```
$smarty->assign("profissao", "professor");
```

```
$smarty->assign("email", "krebistux@gmail.com");
```

Como você pode perceber nos exemplos já apresentados, existe também o método `display`, que provavelmente será utilizado em todas suas aplicações Smarty. Esse é o método responsável por exibir o template. Por exemplo:

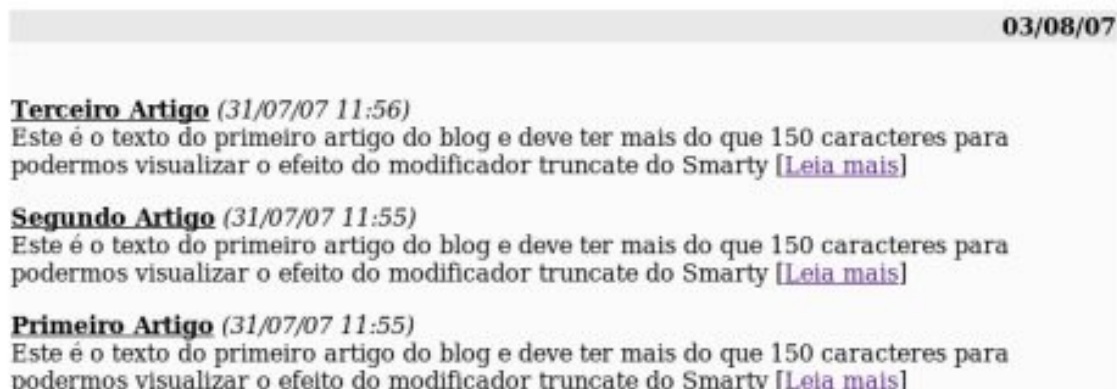
```
$smarty->display("index.tpl");
```

A lista dos métodos da Smarty pode ser encontrada em: http://smarty.php.net/manual/pt_BR/api.functions.php.

9.7 Exemplo de uma aplicação com Smarty

Uma aplicação interessante que podemos criar utilizando templates é um sistema de notícias. Esse sistema pode consultar um banco de dados para exibir sempre as últimas notícias cadastradas, mostrando somente o trecho inicial do texto e

oferecendo ao usuário um link para visualização da notícia completa, como ilustra a figura abaixo.



Para implementar esse sistema, primeiramente vamos criar no MySQL (através do utilitário mysql) uma tabela chamada “noticias”, da seguinte forma:

```
create table noticias
(
    id mediumint NOT NULL AUTO_INCREMENT,
    titulo TEXT NOT NULL,
    texto TEXT NOT NULL,
    data_hora DATETIME NOT NULL,
    primary key(id)
);
```

O campo id será o número que irá identificar a notícia, e posteriormente será passado como parâmetro para a página que irá exibir a notícia completa. Além desse identificador, cada notícia vai possuir um título e um campo para armazenar a data e hora (datetime) de sua publicação.

Esse campo de data e hora será de grande importância, não só para exibir essas informações ao usuário, mas também para ordenar os dados resultantes da consulta SQL. A consulta feita com o comando SQL select irá ordenar os registros resultantes por data e hora de forma decrescente, além de limitar o número de notícias retornadas utilizando a cláusula SQL limit. Dessa forma, serão retornadas apenas as notícias mais atuais.

Acompanhe então como ficaria o programa index.php dessa aplicação. Esse programa faz uma conexão ao banco de dados, obtém as notícias mais atuais, armazena-as em arrays e solicita a exibição do template (que será apresentado logo a seguir).

index.php

```
<?php
//--- acesso ao banco de dados ---//
$servidor = "localhost";
$usuario = "aluno";
$senha = "milenium";
$banco = "curso";
$con = mysql_connect($servidor, $usuario, $senha);
    mysql_select_db($banco);
    $res = mysql_query("select * from noticias order by
data_hora desc limit 3");
    $num_linhas = mysql_numrows($res);
    for($i=0 ; $i<$num_linhas; $i++)
    {
        $array_ids[] = mysql_result($res,$i,"id");
        $array_titulos[] = mysql_result($res,$i,"titulo");
        $array_textos[] = mysql_result($res,$i,"texto");
        $array_datas[] = mysql_result($res,$i,"data_hora");
    }

mysql_close($con);

//--- define variáveis e exibe o template ---//
define('SMARTY_DIR', '/usr/share/php/smarty/libs/');
require_once(SMARTY_DIR . 'Smarty.class.php');

$page = new Smarty;
$page->assign("ids",$array_ids);
$page->assign("titulos",$array_titulos);
$page->assign("textos",$array_textos);
$page->assign("datas",$array_datas);

$page->display("index.tpl");

?>
```

Veja que foram utilizados quatro arrays para armazenar os resultados, sendo um para cada coluna da tabela. Esses valores foram atribuídos a quatro variáveis que serão consultadas no template: \$ids, \$titulos, \$textos e \$datas. Nesse exemplo, esses arrays contêm as três notícias mais atuais existentes em seu banco de dados, visto que utilizamos a cláusula SQL limit com o valor 3.

Para a template a idéia é exibir uma tabela HTML contendo apenas o trecho inicial de cada uma delas (150 caracteres), disponibilizando ao lado um link para que o usuário possa visualizar a notícia completa. Para isso utilizaremos alguns recursos da smarty.

index.tpl

```

<html>
<body>
<div align="center">
<table border="0" cellpadding="0" width="60%">
  <tr>
    <td width="50%" bgcolor="#E8E8E8">
      <p align="right"><b>{$smarty.now|date_format:"%d/%m/
%y"}</b>
    </td>
  </tr>
  <tr>
    <td width="100%" colspan="2" bgcolor="#FBFBFB"><br>
      {section name=i loop=$titulos}
        <p><b><u>{$titulos[i]}</u></b>
        <i>({$datas[i]|date_format:"%d/%m/%y %H:%M"})</i><br>
        {$textos[i]|truncate:150}
        [<a href="mostra_noticia.php?id={$ids[i]}">Leia
mais</a>]</p>
      {/section}
    </td>
  </tr>
</table>
</div>
</body>
</html>

```

Na primeira linha da tabela utilizamos uma variável especial `$smarty` para obter a data atual, juntamente com o modificador `date_format` para exibí-la no formato DD/MM/AAA.

```
{ $smarty.now|date_format:"%d/%m/%y" }
```

Na segunda linha utilizamos a função `section` que funciona como um laço para obter e exibir cada uma das notícias através do array que contém o título das mesmas.

```
{section name=i loop=$titulos}
```

Dentro do laço, utilizamos a variável de controle `"i"`, uma variável do smarty e não do PHP, por isso não temos `"$"` antes do nome da variável. Essa variável é utilizada como índice para acessar os elementos dos quatro arrays (identificadores, títulos, textos e datas).

Para exibir a data e hora de publicação da notícia, foi aplicado o modificador `date_format` para indicar o uso do formato DD/MM/AAA HH:MM.

```
{ $datas[i]|date_format:"%d/%m/%y %H:%M" }
```

Em relação ao texto da notícia, foi usado o modificador truncate para exibir apenas os seus 150 primeiros caracteres. Após exibir o número de caracteres especificado, esse modificador acrescenta automaticamente os três pontos(...) ao final, para indicar que o texto continua.

```
{ $textos[i] truncate:150 }
```

Por fim, foi acrescentado um link ao lado de cada notícia com o texto “Leia Mais”. Esse link chama o programa mostra_noticia.php para exibir a notícia completa e passa como parâmetro o seu identificador.

```
<a href="mostra_noticia.php?id={$ids[i]}">Leia mais</a>
```

A parte da aplicação que exibe uma notícia completa também utiliza templates e segue o mesmo princípio abordado até agora:

mostra_noticia.php

```
<?php
$id_noticia = $_GET['id'];
// --- acesso ao banco de dados --- //
$servidor = "localhost";
$usuario = "aluno";
$senha = "mandriva";
$banco = "curso";
$con = mysql_connect($servidor, $usuario, $senha);
    mysql_select_db($banco);
    $res = mysql_query("select titulo,texto,data_hora from noticias
where id = $id_noticia");
    $num_linhas = mysql_numrows($res);
    if($num_linhas > 0)
    {
        $titulo = mysql_result($res,0,"titulo");
        $texto = mysql_result($res,0,"texto");
        $data = mysql_result($res,0,"data_hora");
    }
    mysql_close($con);

// --- define variáveis e exibe o template --- //
define('SMARTY_DIR', '/usr/share/php/smarty/libs/');
require_once(SMARTY_DIR . 'Smarty.class.php');

$page = new Smarty;
$page->assign("titulo",$titulo);
$page->assign("texto",$texto);
$page->assign("data",$data);
$page->display('mostra_noticia.tpl');
?>
```

mostra_noticia.tpl

```
<html>
<head>
<title>{$titulo}</title>
</head>
</body>
<p>
    <font size="5">{$titulo}</font><br>
    <i>{$data|date_format:"Data: %d/%m/%y Hora: %H:%M"}</i>
</p>
<hr>
<p>{$texto}</p>
<p align="center"><a
href="JavaScript:history.back();">Voltar</a></p>
</body>
</html>
```

Mais funções da Smarty para criação de templates podem ser vistas em:

http://smarty.php.net/manual/pt_BR/language.syntax.functions.php

http://smarty.php.net/manual/pt_BR/language.builtin.functions.php

Mais métodos da classe Smarty para o PHP podem ser vistos em:

http://smarty.php.net/manual/pt_BR/language.custom.functions.php

http://smarty.php.net/manual/pt_BR/api.functions.php

10. Arquivos e Streams

10.1 Introdução

Streams foram introduzidas com o PHP 4.3.0 como um meio para generalizar o acesso a arquivos, sockets, dados comprimidos, entre outros, através do uso de um conjunto comum de funções.

O uso de streams possibilita que os detalhes de acesso para os recursos citados acima sejam abstraídos em diversas camadas utilizadas na implementação de streams, as quais geralmente não precisamos nos preocupar, facilitando, e muito a manipulação destes recursos.

10.2 Acesso a arquivos

Começamos com as funções mais fundamentais que antigamente eram utilizadas somente para manipulação de arquivos, porém a partir do PHP 4.3.0 elas foram estendidas para trabalhar com quase tudo.

fopen()	Abre um handle para um arquivo local, ou um arquivo de uma URL.
fread()	Lê um bloco de dados de um arquivo.
fgets()	Lê uma única linha de um arquivo.
fwrite()/fputs()	Escreve um bloco de dados em um arquivo.
fclose()	Fecha o handle de arquivo aberto.
feof()	Retorna true (verdadeiro) se o fim do arquivo for atingido.

Eis um exemplo de como é simples trabalhar com arquivos em php:

```
<?php
/* Abre um arquivo */
$fp = fopen ( 'data.dat', 'r');
if (!$fp) {
    die ("O arquivo não pode ser aberto.");
}
```

```

/* Lê uma linha do arquivo */
$line = fgets($fp);

/* Fecha o handle do arquivo */
fclose ($fp);
?>

```

A função `fopen` recebe dois parâmetros, o caminho para o arquivo a ser aberto e o modo como o mesmo será aberto. Caso o caminho esteja na forma “protocolo://...” é assumido que seja uma URL e o php irá procurar o manipulador de protocolo conforme o prefixo “protocolo”.

Os modos para abertura de um arquivo são:

r	somente leitura. O ponteiro é colocado no início do arquivo.
r+	leitura e escrita. O ponteiro é colocado no início do arquivo.
w	somente escrita. O arquivo é esvaziado e ponteiro é colocado no início do arquivo.
w+	leitura e escrita. O arquivo é esvaziado e ponteiro é colocado no início do arquivo.
a	somente escrita. O ponteiro é colocado no final do arquivo.
a+	leitura e escrita. O ponteiro é colocado no final do arquivo.

O modificador `b` pode ser utilizado para especificar que o arquivo é binário. Em sistemas UNIX*like o modificador `b` não tem qualquer efeito, apesar disto o seu uso é recomendado quando se está trabalhando com arquivos binários por motivos de portabilidade.

```

<?php
/* Abre um arquivo em modo leitura/escrita e em modo binário, e
posiciona
* ponteiro no início do arquivo. */
$fp = fopen ("/tmp/tempfile", "rb+");
/* Tenta ler um bloco de 4096 bytes do arquivo */
$block = fread($fp, 4096);
/* Escreve esse mesmo bloco de dados de volta para o stream
* logo após o primeiro
fwrite($fp, $block);
/* Fecha o stream */

```

```
fclose ($fp);  
?>
```

Há um terceiro parâmetro opcional, “true”, disponível para `fopen()`, que diz ao PHP para procurar o arquivo no seu caminho include.

Cuidado ao escolher o tamanho máximo das linhas que serão lidas com `fgets()`, o PHP aloca esta memória antes de fazer a leitura, então se você utilizar 1 000 000, o PHP aloca 1Mb de memória, mesmo que sua linha tenha apenas 12 caracteres. O padrão é 1024 bytes, que deve ser suficiente para quase todas as aplicações.

10.3 Input/Output de Programas

Da mesma forma como o UNIX tem o paradigma “Todo IO é um arquivo”, o PHP tem o paradigma “Todo IO é um stream”. Assim, quando quiser trabalhar com o input e o output de um programa você abre um stream para esse programa. Pelo fato de precisar abrir dois canais no seu programa – um para leitura e um para escrita -, você usará duas funções especiais para abrir os streams: `popen()` ou `proc_open()`.

10.3.1 *popen()*

Fornece apenas IO unidirecional a um programa, isto é, você somente poderá utilizar **w** ou **r** como modo de abertura. Quando você abre um stream para um programa, também chamado de pipe (daí o nome `popen()`), você também poderá utilizar todas as funções de arquivos normais para ler ou escrever no pipe, como `feof()` por exemplo, caso não haja mais nenhum input a ser lido. Eis um exemplo que lê o resultado de “ls -l /” :

```
<?php  
$fp = popen('ls -l /', 'r');  
while (!feof($fp))  
{  
    echo fgets($fp). "<br>";  
}  
  
pclose($fp);  
?>
```

10.3.2 *proc_open()*

`popen()` raramente é útil, porque você não pode realizar nenhuma tarefa interativa com o processo aberto. Mas não se preocupe – o PHP tem a função

`proc_open()` para suprir esta necessidade. Com esta função você pode vincular todos os handlers de entrada e saída de um processo a um pipe a partir do qual você pode ler, ou então a um pipe no qual você pode escrever a partir do seu script, ou a um arquivo. Os pipes são tratados como handlers de arquivos, exceto pelo fato de que você nunca poderá abrir um handler de arquivo para leitura e escrita ao mesmo tempo.

`proc_open()` requer três parâmetros:

```
resource proc_open( string cmd, array descriptorspec, array pipes)
```

O parâmetro `cmd` é o comando a ser executado, como por exemplo `/usr/local/bin/php`. O parâmetro `descriptorspec` é mais complexo, `descriptorspec` é um array com cada elemento descrevendo um handler de arquivo para entrada ou saída.

10.3.2.1 Descritores de Arquivos

```
<?php
$fin = fopen("readfrom", "r");
$fout = fopen("writeto", "w");
$desc = array( 0 => $fin, 1 => $fout);
$res = proc_open("php", $desc, $pipes);
if($res) {
    proc_close($res);
}
?>
```

Este script inicia um intérprete PHP – um processo descendente. Ele vincula a entrada para o processo descendente ao descritor de arquivos `$fin` (que é um handler para o arquivo “readfrom”) e a saída do processo descendente a `$fout` (que é um handler para o arquivo “writeto”). O arquivo “readfrom” contém:

```
<?php
echo 'Hello you!';
?>
```

Após a execução do script, o arquivo “writeto” contém:

```
Hello you!
```

10.3.2.2 Pipes

Ao invés de usar um handler de arquivo para entrada e saída para o processo descendente PHP, como mostrado no script da seção anterior, podemos abrir pipes para processos descendentes que nos permitam controlar o processo gerado pelo próprio script. O seguinte script envia `<?php echo 'Hello you!'; ?>` do próprio script para o intérprete PHP gerado. O script escreve o resultado do comando `echo` na saída padrão do script, aplicando `urlencode` à string de texto do resultado, "Hello you!".

```
<?php
$descs = array( 0 => array('pipe', 'r'), 1 => array('pipe', 'w'));
$res = proc_open("php", $descs, $pipes);
if (is_resource($res)) {
    fputs($pipes[0], '<?php echo "Hello you!\n"; ?>');
    fclose($pipes[0]);

    while (!feof($pipes[1])) {
        $line = fgets($pipes[1]);
        echo urlencode($line);
    }

    proc_close($res);
}
?>
```

O resultado é:

Hello+you%21%0A

10.3.2.3 Arquivos

Podemos passar um arquivo como o handler para os descritores de arquivos para o seu processo, como mostrado no seguinte exemplo:

```
<?php
$descs = array(
    0 => array('pipe', 'r'),
    1 => array('file', 'output', 'w'),
    2 => array('file', 'errors', 'w')
);

$res = proc_open("PHP", $descs, $pipes);

if (is_resource($res)) {
    fputs($pipes[0], '<?php echo "Hello you!\n"; ?>');
    fclose($pipes[0]);
    proc_close($res);
}
```



```
}  
?>
```

O arquivo de saída está vazio e o arquivo errors contém:

```
sh: PHP: not found
```

Além do input pipe[0] e do output pipe[1] mostrados nos exemplos, podemos utilizar outros pipes para redirecionar todos os descritores de arquivos do processo descendente. No exemplo anterior, redirecionamos todas as mensagens de erro enviadas ao descritor de erros padrão (2) para o pipe[2], o arquivo errors. O índice do array \$descriptors não se limita aos índices 0-2, de forma que você pode sempre trabalhar com todos os descritores de arquivos da maneira que lhe convier. Entretanto, esses descritores de arquivos adicionais, com um índice maior do que 2, ainda não funcionam no Windows porque o PHP não implementa uma maneira de o processo cliente se anexar a eles. Talvez isso seja solucionado nos desenvolvimentos futuros do PHP.

10.4 Streams de Input/Output

Com o PHP, você pode usar stdin, stdout e stderr como arquivos. Esses “arquivos”, vinculados com os streams stdin, stdout e stderr do processo PHP, podem ser acessados usando-se um especificador de protocolo na chamada a fopen(). Para os streams de input e de output de programas, esse especificador é o `php://`. Essa funcionalidade é útil principalmente ao se trabalhar com a Interface de Linha de Comando (CLI). O seguinte exemplo mostra como obter dados a partir de um formulário que tem dois campos com o mesmo nome:

form.html:

```
<html>  
  <form method="POST" action="process.php">  
    <input type="text" name="example">  
    <select name="example">  
      <option value="1">Example line 1</option>  
      <option value="2">Example line 2</option>  
    </select>  
    <input type="submit">  
  </form>  
</html>
```

process.php

```
<h1>Dumping $_POST</h1>

<?php
    print_r($_POST);
?>

<h1>Dumping php://input</h1>
<?php
    $in = fopen("php://input","rb");
    while(!feof($in)) {
        echo fread($in, 128);
    }
?>
```

O primeiro script contém somente o código HTML para um formulário. O formulário tem dois elementos com o nome “example”: um campo de texto e uma lista de seleção. Quando submetemos o formulário, clicando no botão de submissão, o script process.php executa e exibe o resultado:

Dumping \$_POST

```
Array ( [example] => 2 )
```

Dumping php://input

```
example=rodrigo&example=2
```

Como podemos ver, apenas um elemento – o valor selecionado na lista – é exibido quando lemos o array \$_POST. Entretanto, os dados de ambos os campos aparecem no stream php://input. Nós mesmos podemos fazer o parsing desses dados puros. Embora dados puros possam não ser particularmente úteis como dados POST simples, são úteis para processar requisições WebDAV ou para processar requisições iniciadas por outras aplicações. O stream php://output pode ser usado para escrever para os buffers de output do PHP, o que é essencialmente a mesma coisa de usar echo ou print(). php://stdin e php://input são somente leitura; php://stdout, php://stderr e php://output são somente de escrita.

10.5 Streams de Compressão

O PHP fornece alguns wrappers (agrupadores) para funções de compressão. Anteriormente, era preciso usar funções especializadas para acessar arquivos comprimidos gzip e bzip; agora podemos utilizar o suporte a streaming para essas bibliotecas. Ler e escrever em um arquivo comprimido com gzip ou com bzip funciona exatamente da mesma maneira que ler e escrever em um arquivo normal.

Os streams gzip suportam mais especificadores de modo do que os padrões r, w, a, b e +. Esses modificadores adicionais incluem os níveis de compressão 1 – 9 e os métodos de compressão f para compressão filtrada e h para compressão huffman somente. Esses modificadores somente fazem sentido se você abrir o arquivo para escrita.

No seguinte exemplo, demonstramos como copiar um arquivo de um arquivo bzipado para um arquivo gzipado. Fazemos uso do especificador de nível de compressão 1 para acelerar a compressão, e o terceiro parâmetro fopen(), para especificar a busca pelo arquivo no caminho include. Cuidado ao usar o parâmetro do caminho include, o que torna a execução do seu script mais lenta porque as operações de arquivos são geralmente lentas na maioria dos sistemas operacionais.

```
<?php
ini_set('include_path', '/var/log:/usr/var/log:.');

$url = 'compress.bzip2://logfile.bz2';
$fil = 'compress.xlib://fool.gz';

$fr = fopen($url, 'rb', true);
$fw = fopen($fil, 'wb1');
if( is_resource($fr) && is_resource($fw) ) {
    while(!feof($fr)) {
        $data = fread($fr, 1024);
        fwrite($fw, $data);
    }
    fclose($fr);
    fclose($fw);
}
?>
```

Este script primeiramente define o caminho include como /var/log, /usr/var/log, e o diretório corrente (.). Em seguida, tenta abrir o arquivo logfile.bz2 do caminho include e abre o arquivo fool.gz para escrita com o nível de compressão 1. Se ambos os streams forem abertos com sucesso, o script lê a partir do arquivo bzipado até

chegar ao fim, e escreve o conteúdo diretamente para o arquivo gzipado. Quando o script termina de copiar o seu conteúdo, ele fecha os streams.

Nota: um outro ótimo aspecto dos streams é que você pode aninhar agrupadores. Por exemplo, você pode abrí-los a partir da seguinte URL: `compress.zlib://http://www.example.com/foobar.gz`

10.6 Streams URL

Os streams URL têm um caminho que lembram uma URL, como por exemplo <http://example.com/index.php> ou <ftp://user:password@ftp.example.com>. Na realidade, todos os wrappers especiais usam um caminho no formato de URL, como por exemplo `compress.zip://file.gz`. Entretanto, somente, esquemas que lembrem um recurso remoto, como um arquivo em um servidor FTP ou um documento em um servidor gopher, entram na categoria de streams URL. Os streams URL básicos que o PHP suporta são:

<code>http://.</code>	Para arquivos localizados em um servidor HTTP
<code>https://.</code>	Para arquivos localizados em um servidor HTTP com SSL
<code>ftp://.</code>	Para arquivos em um servidor FTP
<code>ftps://.</code>	Para arquivos em um servidor FTP com suporte a SSL

Para autenticação para servidores HTTP ou FTP, podemos prefixar o hostname na url com `username:password@`, como no seguinte exemplo:

```
$fp = fopen('ftp://derick:secret@ftp.php.net', 'wb');
```

O handler HTTP suporta somente a leitura de arquivos, então você precisa especificar o modo `rb` (estritamente falando o `b` somente é necessário no Windows, mas não faz mal adicioná-lo). O handler de FTP suporta a abertura de um stream somente em modo de leitura ou de escrita, mas não em ambos simultaneamente. Além disso, se você tentar abrir um arquivo existente para escrita, a conexão falha, a não ser que você defina a opção de contexto `'overwrite'`:

```
<?php
$context = stream_context_create( array( 'ftp' => array('overwrite' =>
true ) ) );
$fp = fopen('ftp://secret@ftp.php.net', 'wb', false, $context);
```

```
?>
```

```
<?php

$url = "http://localhost";
$fil = "rodrigo_maluco_com_br.tar.gz";
$begin_time = time();
$size = 0;

$fp = fopen($url, "rb", false);
if( is_resource($fp)) {
    /*Abre o arquivo local*/
    $fs = fopen($fil, "wb9", false);
    if( is_resource($fs)) {
        echo "[ * ";
        /* Lê os dados da URL em blocos de 1024 bytes */
        while(!feof($fp)) {
            $data = fgets($fp, 1024);
            fwrite($fs, $data);
            $size += 1024; //bytes capturados
            usleep(100000); //ctrl de velocidade
            echo "* ";
        }
        $elapsed_time = time() - $GLOBALS['begin_time'];
        /* exibe informações de download */
        printf("] Download time: %ds Speed: %d %s\n",
            $elapsed_time,
            $size/1024/$elapsed_time,
            "Kb/s");

        /* fecha o arquivo local */
        fclose($fs);
    }
    /* fecha o arquivo remoto */
    fclose($fp);
}

?>
```

11. Manipulação de Gráficos com GD

Ao invés de descrever todas as funções GD suportadas pelo PHP, discutiremos dois usos comuns da biblioteca de imagens GD. No primeiro exemplo, usamos as bibliotecas GD para criar uma imagem com um code word nela. Também adicionaremos algumas distorções para que a imagem fique ilegível para a máquina - a proteção perfeita contra ferramentas automáticas que preenchem formulários. No segundo exemplo, criaremos um gráfico de barras, incluindo eixos, rótulos, fundo, texto True Type e alpha blending.

Os exemplos requerem a biblioteca GD instalada.

11.1 Formulários de submissão à prova de Bots

O seguinte script dificulta a submissão de formulários por ferramentas automáticas. Os passos envolvidos neste script básico são criar um espaço de desenho, alocar cores, preencher o fundo, desenhar os caracteres, adicionar distorções e envia a imagem para a saída no navegador:

```
<?php
    $size_x = 200;
    $size_y = 75;
    if( !isset($_GET['code']) ) {
        $code = 'unknow';
    }
    $code = @substr($_GET['code'], 0, 8);
    $space_per_char = $size_x / (strlen($code) + 1);
```

No código anterior, definimos os tamanhos horizontal e vertical das imagens em variáveis, facilitando possíveis modificações futuras. Em seguida, obtemos o código do parâmetro code do array superglobal _GET e o limitamos a um máximo de oito caracteres. Em seguida calculamos \$space_per_char - espaço entre caracteres para uso em uma posterior renderização, no script.

Nota: usar parâmetros de _GET para obter o código obviamente invalida todo o propósito deste script, porque um bot poderia simplesmente ler o arquivo HTML que inclui a linha . Para esse propósito funcionar, precisamos armazenar o código em um banco de dados e, por exemplo, ler, com uma chave aleatória, o código de volta para o script que gera a imagem, mais ou menos como no seguinte exemplo:

```
mysql_connect();  
$res = mysql_query('SELECT code FROM codes WHERE key = ' . (int)  
$_GET['key'] );  
$code = mysql_result($res, 0);
```

e inserí-lo na página HTML com:

```
<img src='image.php?key=90' />
```

```
/* cria canvas */  
$img = imagecreatetruecolor($size_x, $size_y);
```

Com `imagecreatetruecolor()`, criamos uma nova "canvas" onde podemos desenhar usando 256 tons diferentes de vermelho, verde e azul, e um canal alpha por pixel. O PHP fornece uma outra variante de `imagecreate` que pode ser usada para criar "imagens em paleta" com no máximo 256 cores, mas `imagecreatetruecolor()` é mais usada porque as imagens produzidas por ela normalmente têm melhor aparência. Tanto arquivos JPEG como PNG suportam imagens true color, então usaremos essa função para o nosso arquivo PNG. O fundo padrão é de cor preta. Uma vez que queremos mudar o fundo, precisamos "alocar" algumas cores, da seguinte forma:

```
/* Aloca cores */  
$background = imagecolorallocate($img, 255, 255, 255);  
$border = imagecolorallocate($img, 128, 128, 128);  
$colors[] = imagecolorallocate($img, 128, 64, 192);  
$colors[] = imagecolorallocate($img, 192, 64, 128);  
$colors[] = imagecolorallocate($img, 108, 192, 64);
```

No código anterior, usamos `imagecolorallocate()` para definir cinco cores diferentes - `$background`, `$border` e `$colors`, um array contendo três cores a serem usadas na renderização do texto. Em cada chamada à função, passamos a variável `$img` (o recurso de imagem retornado pela função `imagecreatetruecolor()` anteriormente no script), seguida de três parâmetros especificando valores de cores. O primeiro indica a quantidade de tom vermelho, o segundo de tom azul e o terceiro de tom verde na cor final. Os valores de cada tom podem variar de 0 a 255.

```
/* Preenche o fundo */  
imagefilledrectangle($img, 1, 1, $size_x - 2, $size_y - 2, $background);
```

```
imagerectangle($img, 0, 0, $size_x -1, $size_y -1, $border);
```

Usando as duas funções, nós mudamos a cor de fundo para branco e adicionamos a borda cinza. Ambas as funções aceitam os mesmos parâmetros: o recurso da imagem, as coordenadas do canto esquerdo superior, as coordenadas do canto direito inferior e a cor. As coordenadas vão de 0 a `size_x -1`, e de 0 a `size_y -1`, então desenhamos um retângulo preenchido a partir da posição 1, 1 até `size_x - 2`, `size_y - 2`. Desenhamos também uma borda cinza ao redor da imagem.

```
/* desenha o texto */
for( $i=0;$i<strlen($code); $i++ )
{
    $color = $colors[$i % count($colors)];
    imagettftext(
        $img,
        28 + rand(0, 8),
        -20 + rand(0, 40),
        ($i + 0.3) * $space_per_char,
        50 + rand(0, 10),
        $color,
        '/var/www/curso/gd/Vera.ttf',
        $code{$i}
    );
}
```

Neste código, fizemos um loop através de todos os caracteres do código. Primeiramente, escolhemos o próximo elemento no array das cores. Usamos o operador módulo (%) para nos certificarmos de que temos um elemento com essa chave no array. Em seguida, usamos a função `imagettftext()` para desenhar a letra. Passamos os parâmetros mostrados na tabela abaixo para `imagettftext()`.

Parâmetro	Conteúdo	Comentários
<code>img</code>	<code>\$img</code>	O recurso da imagem na qual o desenho será feito.
<code>fontsize</code>	<code>28 + rand(0,8)</code>	O tamanho em pontos (e não em pixels) dos caracteres a serem desenhados. Selecionamos um tamanho aleatório entre 28 e 36 pontos.
<code>angle</code>	<code>-20 + rand(0,40)</code>	O ângulo no qual o caractere será desenhado, em graus (a faixa é de 0 – 360).
<code>x</code>	<code>(\$i + 0,3) * \$space_per_char</code>	A localização x onde o caractere será desenhado (aleatória)
<code>y</code>	<code>50 + rand(0,10)</code>	A localização y onde o caractere será desenhado (aleatória)

colour	\$color	A cor a ser usada para desenhar o texto.
font	'arial.ttf'	O nome do arquivo de fonte a ser usado
text	\$code{\$i}	O caractere do código que desenharemos.

```
/* Adicionando algumas distorções aleatórias */  
  
for ($i = 0; $i < 1000; $i++)  
{  
    $x1 = rand(5, $size_x - 5 );  
    $y1 = rand(5, $size_y - 5 );  
    $x2 = $x1 - 4 + rand(0, 8);  
    $y2 = $y1 - 4 + rand(0, 8);  
    imageline($img, $x1, $y1, $x2, $y2, $colors[ rand( 0, count($colors)  
-1 ) ]);  
}
```

Desenhamos 1000 pequenas linhas com coordenadas aleatórias tanto para o início como para o fim. A função `imageline()` tem os seguintes parâmetros: recurso da imagem, coordenadas x e y iniciais, coordenadas x e y finais, e a cor com a qual desenhar a linha.

```
/* saída para o navegador */  
header('Content-type: image/png');  
imagepng($img);  
?>
```

Ao final do nosso script, usamos a função `header()` para dizer ao navegador que espere dados do tipo imagem png. Este mime-type é associado com uma imagem PNG pelo navegador, de modo que ele saiba como manipular devidamente os dados. Diferentes tipos de dados possuem diferentes tipos mime. Para imagens, você pode especificar `image/gif` (para imagens GIF), `image/jpeg` (para imagens JPEG), `application/octet-stream` (para dados binários), e outros tipos mime. Com o header `Content-type HTTP`, dizemos ao navegador o que esperar.

Nota: a função `header()` somente pode ser utilizada caso nenhum conteúdo tenha sido enviado para a saída antes. Isso inclui espaços em branco, strings vazias... qualquer comando `echo` e/ou `print` utilizado anteriormente resultará em um Warning quando a função `header()` for processada.

Finalmente chamamos a função `imagepng()`, que aceita o recurso da imagem como o seu primeiro parâmetro e possivelmente um nome de arquivo como segundo parâmetro, onde será gravada a imagem.

Nota: Cada tipo de imagem tem uma função específica. Por exemplo: `imagewbmp()`, `imagejpeg()`, etc.

11.2 Gráfico de Barras

O princípio para criar um gráfico de barras através da biblioteca GD é o mesmo visto anteriormente, porém teremos que nos preocupar com algumas novas características, como fundo, barras transparentes e posicionamento preciso do texto.

```
/* pré-definições */
$size_x = 640;
$size_y = 480;
$title = 'Acessos ao site X por ano';
$title2 = 'Acessos (em 1.000)';
define('ARIAL_TTF',
'/usr/share/fonts/truetype/msttcorefonts/arial.ttf');
```

Como no exemplo anterior, primeiramente armazenamos o tamanho horizontal e o vertical da imagem em variáveis. O resto do script calculará a escala correta (exceto para o fundo) se esses valores forem modificados. Para facilitar as coisas, também definimos os títulos no início.

```
$values = array(
    1999 => 5300,
    2000 => 5700,
    2001 => 6400,
    2002 => 6700,
    2003 => 6600,
    2004 => 7100
);

$max_value = 8000;
$units = 500;
```

O array `$values` define o nosso conjunto de dados, a partir do qual desenharemos as barras do nosso gráfico. Normalmente, você não incluiria esses valores diretamente no script. Preferencialmente, os valores viriam de uma outra fonte, como por exemplo um banco de dados. A variável `$max_value` define o valor máximo do gráfico e é usada para calcular a escala dos valores automaticamente. A variável `$units` define a distância entre as linhas verticais da grade.

```
$img = imagecreatetruecolor($size_x, $size_y);
imagealphablending($img, true);
```

A seção anterior do script carrega a imagem de fundo com `imagecreatefrompng()`. Existem funções semelhantes para leitura de arquivos JPEG (`imagecreatefromjpg()`) e GIF (`imagecreatefromgif()`). `getimagesize()` é uma função que retorna um array contendo a largura e a altura de uma imagem, junto com informações adicionais. A largura e a altura são os dois primeiros elementos do array. O terceiro elemento é uma string de texto, `width='640' height='480'`, que você pode inserir em HTML caso necessário. O quarto elemento é o tipo da imagem. O PHP pode determinar o tamanho de cerca de 18 tipos diferentes de arquivos, incluindo PNG, JPEG, GIF, SWF (arquivos flash), TIFF, BMP, e PSD (Photoshop). Com a função `image_type_to_mime_type()`, você pode transformar o tipo no array em um tipo mime válido, como `image/png` ou `application/x-shockwave-flash`.

```
imagecopyresampled(  
    $img, $bg,  
    0, 0, 0, 0,  
    $size_x, $size_y, $sizes[0], $sizes[1]  
);
```

Copiamos o PNG que lemos a partir do arquivo para a imagem de destino – o nosso gráfico. Essa função requer 10 parâmetros. Os dois primeiros são o handle da imagem de destino e o handle da imagem PNG carregada, seguidos por quatro conjuntos de coordenadas: as coordenadas do canto esquerdo superior para a imagem de destino, e as coordenadas do canto direito inferior da imagem fonte. Você pode copiar uma parte da imagem fonte para a imagem de destino usando as coordenadas apropriadas da imagem fonte. A função `imagecopyresized()` também copia imagens e é mais rápida, mas o resultado não é tão bom porque o algoritmo é menos poderoso.

```
/* Área do gráfico */  
  
//azul transparente  
$background = imagecolorallocatealpha($img, 127,127,192,32);  
  
//grafico  
imagefilledrectangle(  
    $img,  
    20, 20 , $size_x - 20, $size_y - 80,  
    $background  
);
```

```
//titulo do grafico
imagefilledrectangle(
    $img,
    20, $size_y - 60, $size_x - 20, $size_y - 20,
    $background
);
```

Desenhemos as duas áreas azuis na imagem de fundo: uma para o gráfico e outra para o título. Uma vez que queremos que as áreas sejam transparentes, criamos uma cor com um valor alpha de 3. O valor alpha deve ser entre 0 e 127, onde zero significa uma cor totalmente opaca e 127 significa totalmente transparente.

```
/* Grade */
$black = imagecolorallocate($img, 0, 0, 0);
$grey = imagecolorallocate($img, 128, 128, 192);
for( $i = $units; $i <= $max_value; $i += $units) {
    $x1 = 110;
    $y1 = $size_y - 120 - (($i / $max_value) * ($size_y - 160));
    $x2 = $size_x - 20;
    $y2 = $y1;
    imageline(
        $img,
        $x1, $y1, $x2, $y2,
        ($i % (2 * $units)) == 0 ? $black : $grey
    );
}

/* Eixos */

imageline($img, 120, $size_y - 120, 120, 40, $black);
imageline($img, 120, $size_y - 120, $size_x - 20, $size_y - 120,
$black);
```

A grade e o eixo são desenhados de forma semelhante. A única coisa que vale a pena mencionar é que colorimos cada segunda linha horizontal de preto, e as outras de cinza.

```
/* Valores */
$barcolor = imagecolorallocatealpha($img, 3, 3, 3, 50);
$spacing = ($size_x - 140) / count($values);

$start_x = 120;

foreach ($values as $key => $value) {
    $x1 = $start_x + 0.2 * $spacing;
    $x2 = $start_x + 0.8 * $spacing;

    $y1 = $size_y - 120;
```

```
$y2 = $y1 - (($value / $max_value) * ($size_y - 160));  
  
imagefilledrectangle($img, $x1, $y2, $x2, $y1, $barcolor);  
$start_x += $spacing;  
}
```

Desenhamos as barras (conforme definidas no array \$values criado no início do script) com imagefilledrectangle(). Calculamos o espaçamento entre as barras dividindo a largura disponível para as barras (largura da imagem menos as margens externos, totalizando 140 – 120 à esquerda e 20 à direita) pelo número de valores do nosso array. O loop incrementa o componente \$start_x na quantia correta e a barra é desenhada de 20% a 80% do seu espaço horizontal disponível. Verticalmente, levamos em conta o valor máximo permitido para o desenho e ajustamos o tamanho de acordo com ele.

```
/* Titulo X */  
$c_x = $size_x / 2;  
$c_y = $size_y - 40;  
  
$box = imagettfbbox( 20, 0, ARIAL_TTF, $title);  
$sx = $box[4] - $box[0];  
$sy = $box[5] - $box[1];  
imagettftext(  
    $img,  
    20, 0,  
    $c_x - $sx / 2, $c_y - ($sy/2),  
    $black,  
    ARIAL_TTF, $title  
);
```

Queremos desenhar o título exatamente no meio da nossa barra azul inferior. Assim, precisamos calcular o espaço exato (o container) requerido pelo nosso texto. Usamos imagettfbbox() para isso. Os parâmetros passados são fontsize, angle, fontfile e text. Esses parâmetros devem ser os mesmos do texto que iremos desenhar depois. A função retorna um array com oito elementos, agrupados de dois em dois, para fornecer as coordenadas dos quatro cantos da caixa contenedora. Os grupos representam o canto esquerdo inferior, o canto direito inferior, o canto direito superior e o canto esquerdo superior.

```
/* Titulo Y */  
$c_x = 50;  
$c_y = ($size_y - 60) / 2;
```

```
$box = imagettfbbox(14, 90, ARIAL_TTF, $title2);
$sx = $box[4] - $box[0];
$sy = $box[5] - $box[1];
imagettftext(
    $img,
    14, 90,
    $c_x - ($sx / 2), $c_y - ($sy / 2),
    $black,
    ARIAL_TTF, $title2
);
```

Fazemos o mesmo em relação ao título para o eixo Y, exceto pelo fato de que usamos um ângulo de 90 graus. O restante do código permanece o mesmo.

```
/* Rótulos */
$c_y = $size_y - 100;
$start_x = 120;

foreach($values as $label => $dummy) {
    $box = imagettfbbox(12, 0, ARIAL_TTF, $label);
    $sx = $box[4] - $box[0];
    $sy = $box[5] + $box[1];
    $c_x = $start_x + (0.5 * $spacing);
    imagettftext(
        $img,
        12, 0,
        $c_x - ($sx / 2), $c_y - ($sy / 2),
        $black,
        ARIAL_TTF, $label
    );

    $start_x += $spacing;
}

$r_x = 100;
for($i = 0; $i <= $max_value; $i += ($units * 2)) {
    $c_y = $size_y - 120 - (($i / $max_value) * ($size_y - 160));

    $box = imagettfbbox(12, 0, ARIAL_TTF, $i / 100);
    $sx = $box[4] - $box[0];
    $sy = $box[5] + $box[1];
    imagettftext(
        $img,
        12, 0,
        $r_x - $sx, $c_y - ($sy / 2),
        $black,
        ARIAL_TTF, $i / 100
    );
}
```

No código acima, desenhamos os diferentes rótulos. Aqueles para o eixo X não são interessantes mas, para os do eixo Y, tentamos alinhar o texto na margem direita não dividindo por 2 a largura do texto a ser desenhado.

```
/* Saída para o navegador */  
header('Content-type: image/png');  
imagepng($img);  
?>
```

12. Expressões Regulares

Embora as expressões regulares sejam bastante poderosas, elas são difíceis de usar, especialmente para iniciantes. Assim, ao invés de mergulhar nas funções que o PHP suporta para lidar com as expressões regulares (a partir de agora chamaremos de “regex”, de regular expression) abordaremos primeiro a sintaxe de combinação de padrões. O PCRE (Perl Compatible Regular Expression) deve estar habilitado, isso pode ser verificado através da saída da função `phpinfo()`.

12.1 Sintaxe

As funções PCRE verificam se uma determinada string de teste coincide com um padrão. A sintaxe dos padrões sempre tem o seguinte formato:

```
<delimitador> <padrão> <delimitador> [<modificadores>]
```

Os modificadores são opcionais. O delimitador separa o padrão dos modificadores. O PCRE usa o primeiro caractere da expressão como delimitador. Você deve usar um caractere que não exista no próprio padrão. Ou pode usar um caractere que existe na sua expressão, mas nesse caso você deverá utilizar `\` como caractere de escape. Tradicionalmente, `/` é usado como delimitador, mas outros delimitadores comuns são `|` ou `@`. Fica à sua escolha. Na maioria dos casos escolheríamos `@`, a não ser que precisássemos fazer a verificação em um e-mail ou padrão semelhante que contenha `@`, nesse caso deveríamos usar `/`. A função PHP `preg_match()` é usada para combinar expressões regulares. O primeiro parâmetro passado a função é o padrão. O segundo parâmetro é a string a ser verificada em relação ao padrão, e é também chamado de sujeito. A função retorna `TRUE` (se o padrão coincide com o sujeito) ou `FALSE` (se o padrão não coincide). Você pode também passar um terceiro parâmetro – o nome de uma variável. O texto combinante é armazenado por referência no array com esse nome. Se você precisar usar o texto combinante, mas apenas quiser saber se houve ou não uma combinação você pode omitir o terceiro parâmetro. Resumindo, o formato é o seguinte, com `$matches` sendo opcional:

```
$result = preg_match($pattern, $subject, $matches);
```


12.1.1 Sintaxe dos Padrões

Na maioria dos sistemas UNIX com a biblioteca PCRE instalada, podemos usar `man pcrepattern` para ler sobre toda a linguagem de combinação de padrões, ou dar uma olhada na (um tanto desatualizada) página de Manual do PHP em <http://www.php.net/manual/en/pcre.pattern.syntax.php>.

12.1.2 Metacaracteres

Abaixo temos uma tabela com os metacaracteres especiais, e a maneira que eles podem ser utilizados para construímos padrões.

Caracter	Descrição
\	O caractere geral de separação. Precisamos dele sempre que quisermos utilizar qualquer um dos metacaracteres no seu padrão, ou o delimitador. A barra invertida pode também ser usada para especificar outros caracteres especiais, os que poderemos observar abaixo.
.	Combina exatamente um caractere, exceto um caractere de nova linha <pre>preg_match('/./', PHP 5, \$matches);</pre> <p><code>\$matches</code> agora contém</p> <pre>Array ([0] => P)</pre>
?	Marca o caractere ou sub-/ verifica padrão precedente (opcional) <pre>preg_match('/PHP.?5/', PHP 5, \$matches);</pre> <p>Isto casa tanto com PHP5 como com PHP 5.</p>
+	Combina o caractere ou sub-padrão precedente uma ou mais vezes. <code>'a+b'</code> verifica <code>'ab'</code> , <code>'aab'</code> e <code>'aaaaaab'</code> , mas não <code>'b'</code> . <code>preg_match</code> também retorna TRUE no exemplo, mas <code>\$matches</code> não contém os caracteres em excesso. <pre>preg_match('/a+b/', 'caaabc', \$matches)</pre> <p><code>\$matches</code> agora contém</p> <pre>Array (</pre>

	<p>[0] => aaab</p> <p>)</p>
*	<p>Combina o caractere precedente zero ou mais vezes. '/de*f' verifica 'df', 'def' e 'deeff'. Novamente, os caracteres em excesso não fazem parte da substring combinada, mas não fazem com que a verificação positiva falhe.</p>
{m} {m,n}	<p>Combina o caractere ou sub-padrão precedente 'm' vezes, caso a variante {m} seja usada, ou de 'm' a 'n' vezes, caso a variante {m,n} seja usada. '/tre{1,2}f/' verifica 'tref' e 'treef', mas não 'treeef'. É possível omitir a parte 'm' ou a parte 'n' da expressão, {m,} para 'm' ou mais vezes, e {,n} para até 'n' vezes.</p>
^	<p>Marca o início do sujeito. '/^ghi/' verifica 'ghik' e 'ghi', mas não 'fghi'</p>
\$	<p>Marca o final do sujeito, a não ser que o último caractere seja uma nova linha '\n'. Nesse caso, a verificação será feita logo antes desse caractere de nova linha. 'Derick\$' verifica "Rethans, Derick" e "Rethans, Derick\n" mas não "Derick Rethans".</p>
[...]	<p>Cria uma classe de caracteres com os caracteres dentro dos colchetes. Você pode usar isso para criar um grupo de caracteres a ser combinado. Usando um hífen dentro da classe, você cria uma faixa de caracteres. Caso deseje usar o hífen como um caractere que faz parte da classe, coloque-o como o último caracteres da classe. O circunflexo '^' tem um significado especial caso seja usado como o primeiro caractere da classe. Nesse caso, ele nega a classe dos caracteres, o que significa que o sujeito não coincide com os caracteres listados.</p> <p>Ex1:</p> <pre>preg_match('/[0-9]+/', 'PHP is released in 2005', \$matches);</pre> <p>\$matches agora contém:</p> <p>Array</p> <pre>([0] => 2005)</pre> <p>Ex2:</p> <pre>preg_match('/[^0-9]+/', 'PHP is released in 2005', \$matches);</pre>

	<p>\$matches agora contém:</p> <pre>Array ([0] => "PHP is released in"</pre> <p>)</p> <p>Repare que \$matches não inclui o ponto do sujeito, porque um padrão sempre é verificado em relação a uma string de caracteres consecutivos. Dentro da classe de caracteres, você não pode usar nenhum dos metacaracteres mencionados nesta tabela, exceto ^ (para negar a classe), - (para criar faixas),] (para finalizar a classe, e \ (para fazer o escape de caracteres especiais).</p>
(...)	<p>Cria um sub-padrão, que pode ser usado para agrupar certos elementos em um padrão. Por exemplo, se tivermos a string 'PHP in 2005' e quisermos extrair o século e o ano como duas entradas separadas, usaríamos o seguinte:</p> <pre>\$matchesregexp = '/([12][0-9])([0-9]{2})/'</pre> <p>Isso cria dois padrões:</p> <p>'/[12][0-9]/' para verificar todos os séculos, de 10 a 29</p> <p>'/[0-9]{2}/' para verificar o ano do século.</p> <pre>preg_match(\$matchesregex, 'PHP in 2005', \$matches);</pre> <p>\$matches agora contém:</p> <pre>Array ([0] => 2005 [1] => 20 [2] => 05</pre> <p>)</p> <p>O elemento com o índice 0 é sempre a string que coincide perfeitamente, e todos os sub-padrões recebem um número na ordem em que ocorrerem no padrão.</p>
(?:...)	<p>Cria um sub-padrão que não é capturado no resultado. Você pode usar isto para certificar-se de que o padrão é seguido por algo.</p>

	<pre>preg_match('@([A-Za-z]+)(?:hans)@', 'Derick Rethans', \$matches);</pre> <p>\$matches agora contém:</p> <pre>Array ([0] => Rethans [1] => Ret)</pre> <p>Como você pode ver, a string que coincide perfeitamente ainda inclui a parte completa do sujeito, mas só há um elemento extra para as combinações de sub-padrões. Sem o ?: no segundo sub-padrão, teria havido também um elemento contendo hans.</p>
<?P<name>...)	<p>Cria um sub-padrão nomeado. é o mesmo que um sub-padrão normal, mas gera elementos adicionais no array \$matches.</p> <pre>preg_match('(P<century>[12][0-9])(P<year>[0-9]{2})/', 'PHP in 2005', \$matches);</pre> <p>\$matches agora contém</p> <pre>Array ([0] => 2005 [century] => 20 [1] => 20 [year] => 05 [2] => 05)</pre> <p>Isto é útil caso você tenha um padrão complexo e não queira se dar ao trabalho de descobrir o número de índice correto no array \$matches.</p>

12.2 Exemplo

```
$pattern = '/^([0-9a-f][0-9a-f:]){5}[0-9a-f][0-9a-f]$/';
```

Este padrão verifica um endereço MAC – um número único, vinculado a uma placa de rede – com o formato 00:04:23:7c:5d:01. O padrão é vinculado ao início e ao fim da nossa string sujeito com ^ e \$, e contém duas partes:

`([0-9a-f][0-9a-f:]){5}` Verifica os cinco primeiros grupos de 2 caracteres seguidos por :

`[0-9a-f][0-9a-f]` O sexto grupo de dois dígitos

Esta regex poderia também ter sido escrita como `/^[0-9a-f]{2:}{5}[0-9a-f]{2})$/`, que teria sido um pouco mais curto. Para verificar o texto em relação ao padrão, use o seguinte código:

```
preg_match($pattern, '00:04:23:7c:5d:01', $matches);
echo "<pre>";
print_r($matches);
echo "</pre>";
```

Com qualquer um dos patterns o resultado seria o mesmo:

```
Array
(
    [0] => 00:04:23:7c:5d:01
    [1] => 5d:
)
```

12.3 Funções

Temos disponíveis três grupos de funções relacionadas ao PCRE: funções de combinação, funções de substituição, e funções de divisão. `preg_match()`, discutida anteriormente, pertence ao primeiro grupo. O segundo grupo contém funções que substituem substrings, que coincidem com um padrão específico. O último grupo de funções divide strings (`preg_split()`) com base em combinações de expressões regulares.

12.3.1 Funções de combinação

`preg_match()` é a função que combina um padrão com a string sujeito e retorna ou true ou false, dependendo se o sujeito coincidiu com o padrão ou não. Pode

retornar também um array contendo o conteúdo das diferentes combinações dos subpadrões. A função `preg_match_all()` é semelhante, exceto pelo fato de que ela combina o padrão com o sujeito repetidamente. Encontrar todas as combinações é útil ao se extrair informações de documentos.

12.3.2 Funções de substituição

Além das combinações descritas na seção anterior, as funções das expressões regulares do PHP podem também substituir textos com base em combinações de padrões. As funções de substituição podem substituir uma substring que coincida com um subpadrão por um texto diferente. Na substituição, podemos nos referir as combinações de padrões usando referências passadas. Eis um exemplo que explica as funções de substituição. Neste exemplo, usamos `preg_replace()` para substituir um pseudo-link, como `[link url="www.php.net"]PHP/link]`, por um link HTML real:

```
<?php
$str = '[link url="url="http://php.net"]PHP[/link] is cool.';
$pattern = '@\[link\ url="([^\"]+)"\](.*?)\[\/link\]@';
$replacement = '<a href="\1">\2</a>';
$str = preg_replace($pattern, $replacement, $str);
echo $str;
?>
```

O resultado do script é:

```
<a href="http://php.net">PHP</a> is cool.
```

O padrão consiste de dois sub-padrões, `([^\"]+)` para a URL e `(.*?)`. Ao invés de retornar a substring do sujeito que combina os dois sub-padrões, o engine PCRE atribui a substring a referências passadas, que você pode acessar usando `\1` e `\2` na string de substituição. Se não quiser usar `\1`, você pode usar `$1`. Cuidado ao colocar a string de substituição dentro de aspas duplas, porque você terá de fazer a diferenciação das barras (de modo que uma referência passada se parecerá com `\\\\1`) ou do cifrão (`\\$1`). Você deve sempre colocar a string de substituição dentro de aspas simples. A combinação completa do padrão é atribuída à referência passada 0, da mesma forma como o elemento com a chave no array `matches` da função `preg_match()`.

Nota: se a string de substituição precisar ter a referenciada passada seguida de algum número, podemos utilizar `${1}1`, para a primeira referência passada seguida pelo número 1.

Apêndice A - IDEs

Editores de textos são ótimos, porém nenhum deles possibilita que você desenvolva de uma forma tão eficiente como quando utilizando um IDE. Em particular, não podemos esperar da maioria dos editores de textos, visões estruturadas, diversos arquivos e diretórios vistos em árvores, navegação entre suas funções, classes, métodos, autocompletar códigos, dicas de variáveis e funções,... entre muitos outros recursos que somente um IDE pode nos oferecer.

O que é um IDE?

Do inglês *Integrated Development Environment* ou Ambiente Integrado de Desenvolvimento, é um programa que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo.

Um bom IDE deveria conter um editor, geralmente com suporte a highlighting, um debugador, visualizador de código em árvore (geralmente integrado) e um plugin para algum tipo de gerenciador de versões, como CVS ou subversion (SVN), entre outras funcionalidades. Porém nem todos os IDE's possuem todos estes recursos, alguns possuem muito mais, outros menos.

EasyEclipse

Num ambiente software livre (SL)/código aberto (CA), podemos contar com diversos IDE's para a linguagem PHP como PHP Eclipse, PHP IDE, Zend Studio, PHPedit, Komodo, etc. Porém o único livre de qualquer tipo de encargos, tanto para o desenvolvimento particular como para o desenvolvimento comercial, é o Eclipse, nas suas duas variantes, PHPEclipse e PHP IDE.

A diferença entre eles é a forma como foram construídos e como integram com o ambiente base, o eclipse.

O PHP IDE é um projeto mais recente e ainda não possui uma versão estável, além de possuir uma documentação extremamente escassa. Por esses motivos iremos utilizar o PHP eclipse, empacotado pelo projeto EasyEclipse (www.easyeclipse.org).

Nota: apesar das características negativas sobre o PHP IDE citadas acima, ele segue as recomendações dos padrões do Eclipse, e provavelmente ele se tornará um IDE mais poderoso e estável do que o PHP Eclipse.

Alternativamente poderíamos instalar o eclipse e todos os plugins necessários para desenvolver em PHP manualmente, porém ao utilizar o pacote do EasyEclipse podemos contar com uma melhor integração entre o eclipse e os plugins necessários do que se tivéssemos instalados todos manualmente, além disso não teremos nenhum tipo de dor de cabeça, para resolver os problemas de compatibilidades entre o eclipse, os plugins e as versões. A única desvantagem é ficar dependente do projeto EasyEclipse para obter novas versões dos plugins utilizados.

Para fazer o download do software acesse o link:
www.easyeclipse.org/site/distributions/php.html

Há uma versão específica para o Linux, e para utilizar é só fazer o download e executar o aplicativo “eclipse” dentro da pasta que você deverá descompactar em alguma partição do seu disco rígido.

Nota: costuma-se instalar os softwares de terceiros no diretório /opt. E no curso seguiremos este padrão, extraindo o conteúdo do arquivo compactado do projeto EasyEclipse para o diretório /opt do sistema.

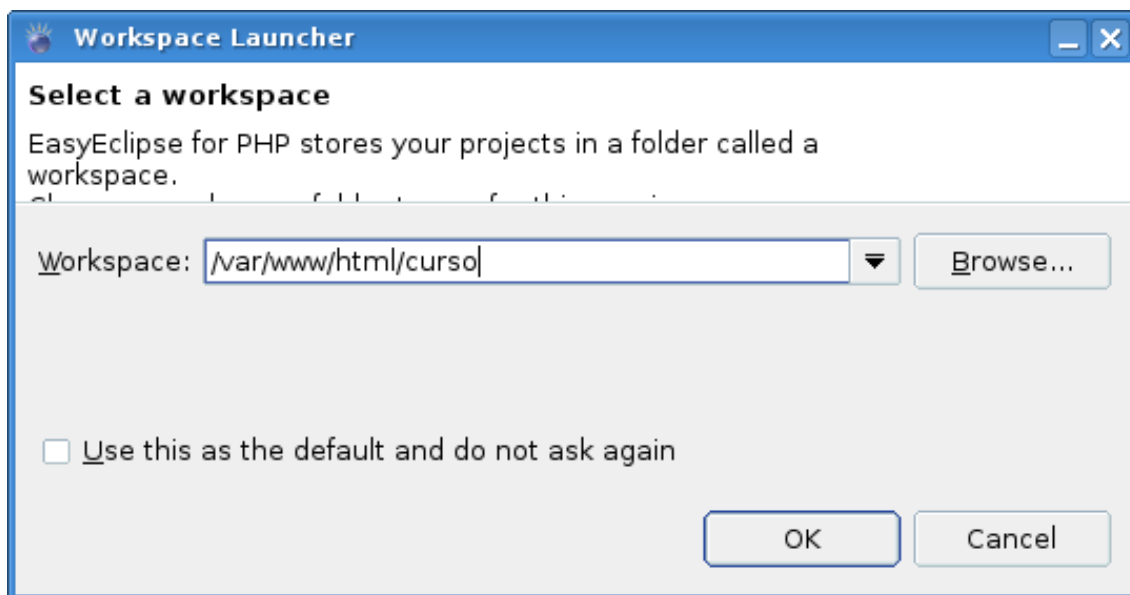
Assumindo que você tenha feito a instalação da release 1.2.2 (que é a mais recente, conforme pesquisa em agosto/07) o programa deverá ser acessado em:

`/opt/easyeclipse-php-1.2.2/eclipse`

Antes de continuarmos é necessário uma plataforma LAMP (Linux, Apache, MySQL e PHP) instalada para desenvolvimento local, ou desenvolver os scripts remotamente, caso estejamos acessando o diretórios dos scripts através de um diretório compartilhado, um ftp ou ssh com o servidor Web.

A interface do EasyEclipse

Quando abrimos o easyeclipse pela primeira vez, nos é apresentada a seguinte tela:

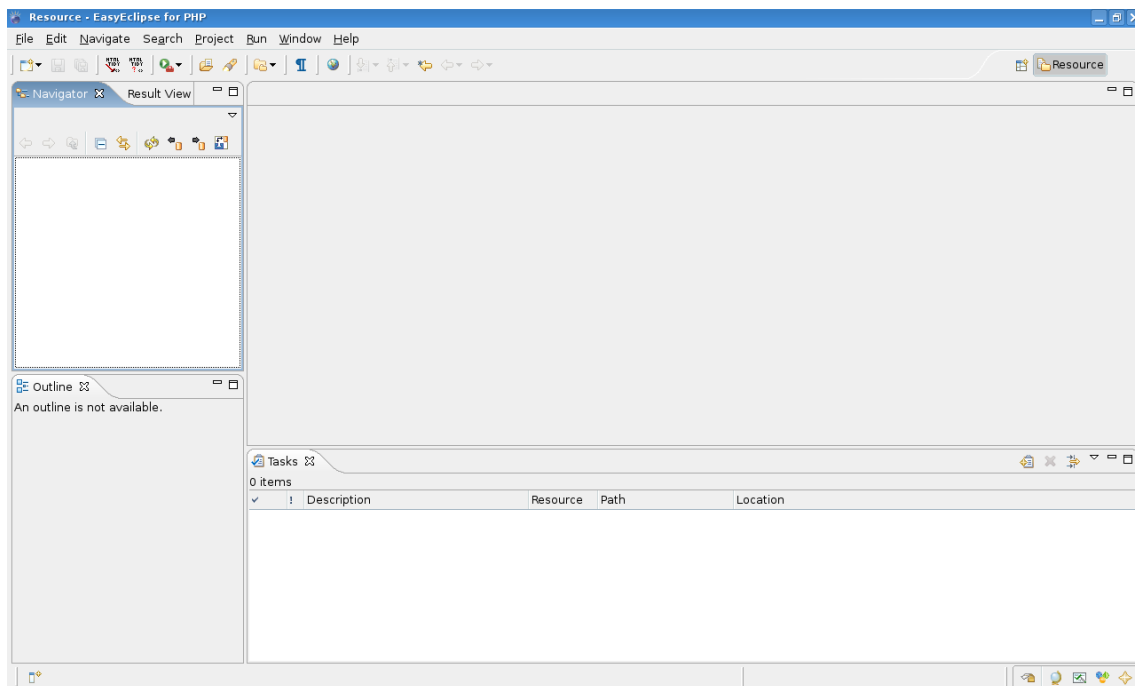


O EasyEclipse quer saber em qual diretório ele deverá salvar os projetos criados e mantidos através dele. Podemos utilizar o diretório indicado ou escolher um diretório que esteja dentro do `DocumentRoot` do servidor web, desde que tenhamos permissão para isso. Para o curso iremos utilizar o diretório: `/var/www/html/curso`

Certifique-se de que o seu usuário tenha permissão para ler e gravar neste diretório, e também que o usuário do servidor Web tenha permissão de leitura para os arquivos e execução nos diretórios.

Na primeira vez que acessamos o EasyEclipse, ele nos disponibiliza uma janela chamada “Welcome”, onde podemos ter uma visão geral do programa, ver quais são as novidades, ver exemplos, seguir um tutorial, ou ir para a área de trabalho clicando no ícone com um seta.

Clicando no ícone da seta teremos acesso ao Workbench:

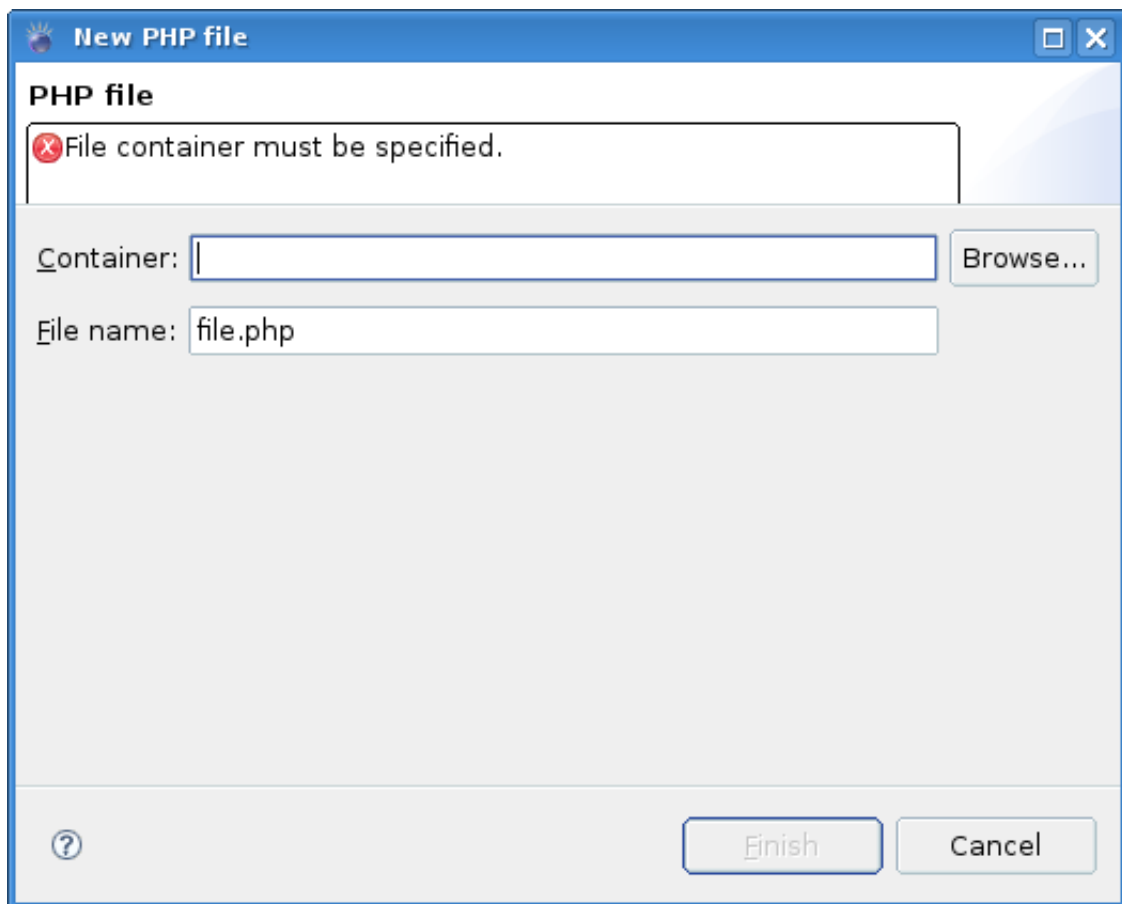


A primeira coisa a fazer é tornar este ambiente mais propício para o desenvolvimento em php. Para isso podemos abrir a “perspectiva” PHP, através do menu Window -> Open Perspective -> Other. Procure pela opção PHP e clique em OK.

Antes de sair criando qualquer tipo de código, precisamos criar um projeto para colocar este código nele. Na barra de ferramentas principal, o primeiro botão (Create) nos dá acesso a opção “PHP Project”, desde que cliquemos na seta que aponta para baixo, e não no botão em si. Escolha um nome para o seu projeto e clique em Finish.

Na área a direita do Workbench, temos acesso ao “Navigator”, onde podemos visualizar o projeto criado, na forma de um diretório, e iremos visualizar todos os arquivos e subdiretórios do nosso projeto.

Vamos criar um arquivo com um código simples, apenas para demonstrar algumas situações comuns. No menu “Create”, o mesmo utilizado para criar o projeto, podemos ter acesso a criação de um novo arquivo “PHP File”, da mesma forma que fizemos para criar um novo projeto.



Devemos informar qual é o “Container”, que nada mais é do que o diretório no qual o arquivo deverá ser salvo, e o nome do novo arquivo.

Após criado o arquivo o EasyEclipse nos apresenta uma janela, com um conteúdo pré determinado, que é incluído automaticamente em todos os novos arquivos. Iremos inserir o seguinte código:

```
<?php
    phpinfo();
?>
```

Quando salvamos este arquivo o EasyEclipse automaticamente executa-o e nos apresenta a saída em uma janela abaixo da janela onde estamos editando o arquivo.

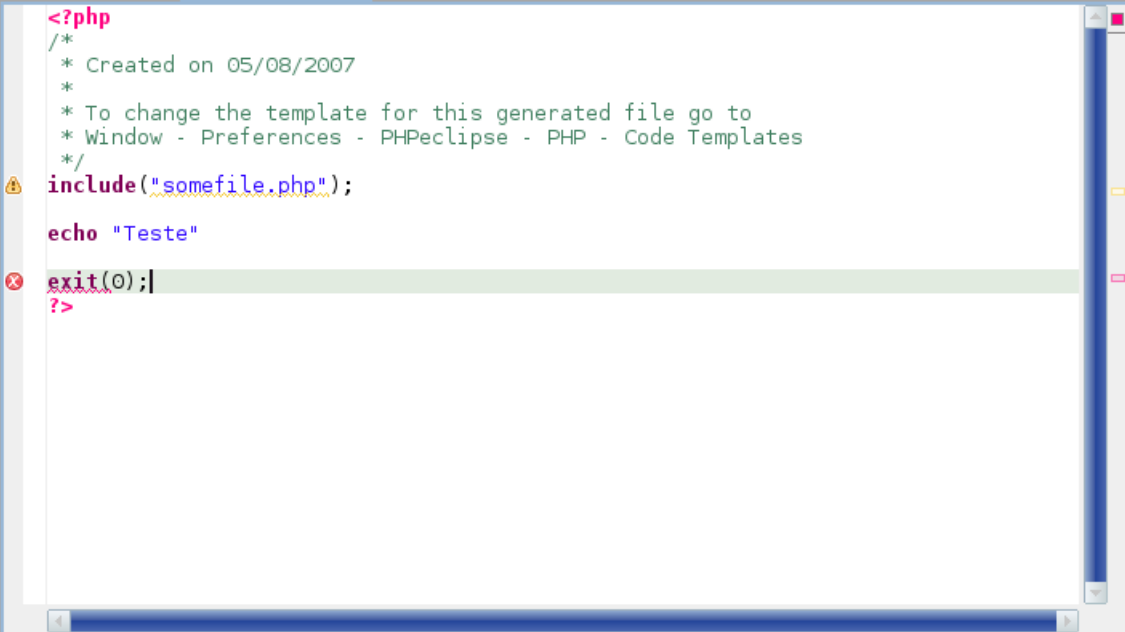
Provavelmente, a primeira vez que ele tentar executar o arquivo ele não conseguirá encontrar corretamente o path para o arquivo em questão. Para resolver este problema temos que editar algumas propriedades do nosso projeto. Para isto, no menu “Navigator” clique com o botão direito sobre a pasta principal do projeto, e acesse a entrada “Properties”.

Na janela que se abre, procure pelo menu “PHP Project Settings”. No contexto que será mostrado, clique em “Configure Workspace Settings”. E altere a entrada “DocumentRoot” que deve estar incorreta (por estar incluindo o diretório do projeto). Para o DocumentRoot configurado para o seu servidor web, provavelmente /var/www/html . A partir de agora sempre que fizermos qualquer alteração no arquivo, e salvarmos estas modificação, automaticamente será exibido o resultado da execução do arquivo na janela abaixo da janela do código do arquivo.

Para continuar insira o seguinte código no seu novo arquivo:

```
<?php
include("somefile.php");
echo "Teste"
exit(0);
?>
```

Perceba que o além de o arquivo “somefile.php” não existir, temos a falta do finalizador de instruções “;” na linha `echo "Teste"`. Ao salvarmos este arquivos o EasyEclipse automaticamente nos mostra as linhas que contém erros ou avisos, colocando um ícone ao lado das mesmas. Para saber qual é o erro ou o aviso da linha, devemos posicionar o cursor sobre o ícone que representa o erro ou o aviso e nos será mostrado um diálogo com o mesmo. A partir do momento que as falhas forem corrigidas os erros e avisos deixarão de ser exibidos. Isso nos poupa muito tempo, tendo que ir no browser para verificar os erros, ou tendo que acessar os arquivos de logs.



```
<?php
/*
 * Created on 05/08/2007
 *
 * To change the template for this generated file go to
 * Window - Preferences - PHPeclipse - PHP - Code Templates
 */
include("somefile.php");

echo "Teste"

exit(0);|
?>
```

The image shows a screenshot of a code editor window, likely from the PHP Eclipse IDE. The editor contains a PHP template file. The code starts with a PHP opening tag, followed by a multi-line comment block. The comment block contains the date '05/08/2007' and instructions on how to change the template for this generated file, pointing to the 'Window - Preferences - PHPeclipse - PHP - Code Templates' path. Below the comment, there is a line of code: `include("somefile.php");`. This is followed by another line of code: `echo "Teste"`. The final line of code is `exit(0);|`, where the vertical bar indicates the current cursor position. The editor window has a standard Windows-style title bar and a vertical scrollbar on the right side.