Autores:
Alexandre Arroyo
Fabio Santos

DTTEC
Centro de Computação
Unicamp

Licenciamento de Uso

Índice

Última atualização em 18/05/2009

Capítulo 1

Programação Orientada à Objetos (POO) com PHP

A orientação à objetos é uma maneira de programar que modela os processos de programação de uma maneira próxima à realidade, tratando a cada componente de um programa como um objeto, com suas características e funcionalidades. O tratamento de objetos no PHP 5 foi totalmente reescrito, permitindo a utilização de uma maior quantidade de recursos da POO, melhor performance e mais vantagens na implementação deste tipo de programação.

A POO também possibilita uma maior otimização e reutilização do código.

1. Classes e Objetos

Classe é a estrutura mais fundamental para a criação de um objeto. Uma classe nada mais é do que um conjunto de variáveis (propriedades ou atributos) e funções (métodos), que definem o estado e o comportamento do objeto. Quando criamos uma classe, temos como objetivo final a criação de objetos, que nada mais são do que representações dessa classe em uma variável. Para declararmos uma classe, utilizamos a palavra-chave *class*. Nosso exemplo será com a classe *Noticia*:

Exemplo 1.1 – Classe Noticia:

```
<?php

# noticia.class.php

class Noticia
{
      public $titulo;
      public $texto;

      function setTitulo($valor)
      {
            $this->titulo = $valor;
      }

      function setTexto($valor)
      {
            $this->texto = $valor;
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$not = new Noticia;
$not->titulo = 'Novo curso de PHP Avançado';
$not->texto = 'Este curso contém os seguinte tópicos: POO, XML, etc.';
$not->exibeNoticia();

?>
```

Obs: como os atributos são do tipo *public*, podemos atribuir valores diretamente para eles, sem a necessidade de utilizar os métodos. Para manipularmos variáveis na classe, precisamos usar a variável *$this*, funções e o separador **->**. A classe deve utilizar a variável *$this* para referenciar seus próprios métodos e atributos.

1.1– Herança

Herança é uma forma de reutilização de código em que novas classes são criadas a partir de classes existentes, absorvendo seus atributos e comportamentos, e complementando-os com novas necessidades. O exemplo da vez será com a classe NoticiaPrincipal:

Exemplo 1.1.1 – Teste de Herança com a Classe NoticiaPrincipal:

```
<?php

# noticia_heranca.php

include_once('noticia.class.php');

class NoticiaPrincipal extends Noticia
{
      public $imagem;

      function setImagem($valor)
      {
            $this->imagem = $valor;
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "<img src=\"". $this->imagem ."\"><p>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$not = new NoticiaPrincipal;
$not->titulo = 'Vestibular da Unicamp termina nesta quarta-feira';
$not->texto  = 'Um dos maiores vestibulares do país acaba nesta quarta-feira,';
$not->texto .= 'com número recorde de inscritos';
$not->imagem = 'img_unicamp.jpg';
$not->exibeNoticia();

?>
```

Como mostra o exemplo, a classe NoticiaPrincipal herdou todas as características da classe Noticia, e ainda foi adicionado o método que dá suporte à exibição de imagens nas notícias principais. Nessas sub-classes é possível redefinir métodos, podendo modificá-los da forma que o programador quiser, como foi o caso do método `exibeNoticia()`. Sobrescrever métodos é algo bastante comum no processo de herança, visto que os métodos que foram criados na classe “pai” não precisam ser necessariamente os mesmos que os definidos nas classes “filhas”.

1.2– Método Construtor e Destrutor

- Construtor : É um método que utiliza o nome reservado `__construct()` e que não precisa ser chamado da forma convencional, pois é executado automaticamente quando instanciamos um objeto a partir de uma classe. Sua utilização é indicada para rotinas de inicialização. O método construtor se encarrega de executar as ações de inicialização dos objetos, como por exemplo, atribuir valores a suas propriedades.

Exemplo 1.2.1 – Método construtor na Classe Noticia:

```
<?php

# noticia_construct.class.php

class Noticia
{
      public $titulo;
      public $texto;

      function __construct($valor_tit, $valor_txt)
      {
            $this->titulo = $valor_tit;
            $this->texto  = $valor_txt;
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$not = new Noticia('Novo curso de PHP Avançado','Abordaremos: POO, XML, etc.');
$not->exibeNoticia();

?>
```

E como ficaria a classe filha NoticiaPrincipal com um método construtor?

Exemplo 1.2.2 – Método construtor na subclasse NoticiaPrincipal:

```
<?php

# noticia_construct_heranca.php

include_once('noticia_construct.class.php');

class NoticiaPrincipal extends Noticia
{
      public $imagem;

      function __construct($valor_tit, $valor_txt, $valor_img)
      {
            parent::__construct($valor_tit, $valor_txt);
            $this->imagem = $valor_img;
      }


      function exibeNoticia()
      {
            echo "<center>";
            echo "<u><img src=\"". $this->imagem ."\"></u><p>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$titulo = 'Vestibular da Unicamp termina nesta quarta-feira';
$texto  = 'Um dos maiores vestibulares do país acaba nesta quarta-feira..';
$imagem = 'img_unicamp.jpg';

$not = new NoticiaPrincipal($titulo, $texto, $imagem);
$not->exibeNoticia();

?>
```

O método construtor da classe Noticia é herdado e executado automaticamente na subclasse NoticiaPrincipal. Porém, as características específicas de NoticiaPrincipal não serão inicializadas pelo método construtor da classe pai. Outro detalhe importante: caso a subclasse NoticiaPrincipal tenha declarado um método construtor em sua estrutura, este mesmo método da classe Noticia não será herdado. Nesse caso podemos chamar o método construtor da Classe Noticia, através de uma chamada específica: parent::__construct()

Destrutor : O método destrutor será chamado assim que todas as referências a um objeto em particular forem removidas ou quando o objeto for explicitamente destruído. O método __destruct é executado toda vez que um objeto da classe é destruído pelo fim do script ou através da função unset. Sua função é basicamente zerar variáveis, limpar dados de sessões, fechar conexões com banco de dados, etc.

Como no método construtor, o método destrutor possui um nome reservado, o `__destruct()`. O exemplo a seguir aplica este conceito na classe Noticia e NoticiaPrincipal:

Exemplo 1.2.3 – Método destrutor:

```
function __destruct()
{
    echo “Destruindo objeto...”;
}
```

1.3- Encapsulamento

Este recurso possibilita ao programador restringir ou liberar o acesso às propriedades e métodos das classes. A utilização deste recurso só se tornou possível à partir do PHP 5. Aplica-se este conceito através dos operadores:

- **Public** : Quando definimos uma propriedade ou método como `public`, estamos dizendo que suas informações podem ser acessadas diretamente por qualquer script, a qualquer momento. Até este momento, todas as propriedades e métodos das classes que vimos foram definidas dessa forma.

- **Protected** : Quando definimos em uma classe uma propriedade ou método do tipo `protected`, estamos definindo que ambos só poderão ser acessados pela própria classe ou por seus herdeiros, sendo impossível realizar o acesso externo.

- **Private** : Quando definimos propriedades e métodos do tipo `private`, só a própria classe pode realizar o acesso, sendo ambos invisíveis para herdeiros ou para classes e programas externos.

A utilização destes modificadores de acesso encontra-se nos próximos dois exemplos. O primeiro exemplo redefiniu as classes Noticia e NoticiaPrincipal:

Exemplo 1.3.1 – Utilização dos modificadores de acesso nas Classes Noticia e NoticiaPrincipal:

```
<?php

# noticia_encapsula.class.php

class Noticia
{
      protected $titulo;
      protected $texto;

      function setTitulo($valor)
      {
            $this->titulo = $valor;
      }

      function setTexto($valor)
      {
            $this->texto = $valor;
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

class NoticiaPrincipal extends Noticia
{
      private $imagem;

      function setImagem($valor)
      {
            $this->imagem = $valor;
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "<img src=\"". $this->imagem ."\"><p>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}
?>
```

No segundo exemplo, a classe NoticiaUltimaHora é criada, e herda as características da classe NoticiaPrincipal.

Exemplo 1.3.2 – Utilização dos modificadores de acesso na Classe NoticiaUltimaHora:

```
<?php

# noticia_ultimahora.php

include_once ('noticia_encapsula.class.php');

class NoticiaUltimaHora extends NoticiaPrincipal
{

      function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$titulo = 'Vestibular da Unicamp';
$texto  = 'Um dos maiores vestibulares do país tem número recorde de inscritos';
$imagem = 'img_unicamp.jpg';

$not_princ = new NoticiaPrincipal;
$not_princ->setTitulo($titulo);
$not_princ->setTexto($texto);
$not_princ->setImagem($imagem);
$not_princ->exibeNoticia();

echo "<pre>";
print_r($not_princ);
echo "</pre>";


$titulo = 'Campus recebe 500 novas árvores';
$texto  = 'Plantio, realizado entre 2007 e o primeiro semestre de 2008,';
$texto .= ' integra programa ambiental que estimula a participação da comunidade';

$not_ulthora = new NoticiaUltimaHora;
$not_ulthora->setTitulo($titulo);
$not_ulthora->setTexto($texto);
$not_ulthora->exibeNoticia();

echo "<pre>";
print_r($not_ulthora);
echo "</pre>";

?>
```

1.4- Interfaces

Interfaces permitem a criação de código que especifica quais métodos uma classe deve implementar, sem ter que definir como esses métodos serão tratados. Interfaces são definidas utilizando a palavra-chave `interface`, e devem ter definições para todos os métodos listados na interface. Classes podem implementar mais de uma interface se desejarem, listando cada interface separada por um espaço.

Dizer que uma classe implementa uma interface e não definir todos os métodos na interface resultará em um erro fatal exibindo quais métodos não foram implementados. Vamos criar uma interface que servirá de base para as nossas classes de notícias utilizadas até agora:

Exemplo 1.4.1 – Interface iNoticia:

```
<?php

# noticia_interface.class.php

interface iNoticia
{
      public function setTitulo($valor);
      public function setTexto($valor);
      public function exibeNoticia();
}

?>
```

A implementação desta interface é simples. No código a seguir utilizamos a nossa interface iNoticia para a classe Noticia e conseqüentemente NoticiaPrincipal:

Exemplo 1.4.2 – Interface iNoticia implementada na Classe Noticia:

```
<?php

# noticia_interface_impl.php

include_once('noticia_interface.class.php');

class Noticia implements iNoticia
{
      protected $titulo;
      protected $texto;

      public function setTitulo($valor)
      {
            $this->titulo = $valor;
      }

      public function setTexto($valor)
      {
            $this->texto = $valor;
      }

      public function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}


$titulo = 'Vestibular da Unicamp';
$texto  = 'Um dos maiores vestibulares do país tem número recorde de inscritos';

$not = new Noticia;
$not->setTitulo($titulo);
$not->setTexto($texto);
$not->exibeNoticia();

echo "<pre>";
print_r($not);
echo "</pre>";

?>
```

1.5 - Classes Abstratas

Classes abstratas são classes que não podem ser instanciadas diretamente, sendo necessária a criação de uma sub-classe para conseguir utilizar suas características. Isso não quer dizer que os métodos destas classes também precisem ser abstratos, isso é opcional. Já as propriedades não podem ser definidas como abstratas.

Aqui vemos o conceito de polimorfismo, ou seja, a possibilidade de dois ou mais objetos executarem a mesma ação. Um exemplo prático seria uma moto e um carro, os dois tem a ação em comum de Frear e Acelerar; em orientação a objetos usamos classes abstratas para dar funcionalidades iguais a objetos diferentes.

Vamos transformar a classe Noticia em uma classe abstrata e depois herdar suas características para a sub-classe NoticiaPrincipal. Abaixo, a seqüência de exemplos para demonstrar este recurso:

Exemplo 1.5.1 – Classe Abstrata Noticia

```
<?php

# noticia_abstrata.class.php

abstract class Noticia
{
      protected $titulo;
      protected $texto;

      public function setTitulo($valor)
      {
            $this->titulo = $valor;
      }

      abstract public function setTexto($valor);
      abstract public function exibeNoticia();
}

?>
```

O exemplo acima nos mostra a utilização tanto de métodos abstratos quanto de métodos comuns. Os métodos abstratos não devem conter código, apenas definição. Quando criamos um método abstrato, fazemos com que ele seja implementado em todas as classes que herdarem dessa classe abstrata.
No exemplo a seguir, os métodos abstratos serão definidos na sub-classe que herdará Noticia, ou seja, NoticiaPrincipal:

Exemplo 1.5.2 – Sub-classe NoticiaPrincipal utilizando a classe abstrata Noticia:

```
<?php

# noticia_abstrata.php

include_once('noticia_abstrata.class.php');

class NoticiaPrincipal extends Noticia
{
      private $imagem;

      public function setTexto($valor)
      {
            $this->texto = $valor;
      }

      function setImagem($valor)
      {
            $this->imagem = $valor;
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "<img src=\"". $this->imagem ."\"><p>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$titulo = 'Vestibular da Unicamp';
$texto  = 'Um dos maiores vestibulares do país chega ao fim,';
$texto .= 'com número recorde de inscritos';
$imagem = 'img_unicamp.jpg';

$not = new NoticiaPrincipal;
$not->setTitulo($titulo);
$not->setTexto($texto);
$not->setImagem($imagem);
$not->exibeNoticia();

echo "<pre>";
print_r($not);
echo "</pre>";

?>
```

1.6- A palavra-chave 'final'

Classes definidas com a palavra-chave `final` não podem ser herdadas por outras classes. Quando um método é definido dessa forma, as subclasses que o herdarem não poderão sobrescrevê-los. Os próximos exemplos mostram a utilização deste recurso:

Exemplo 1.6.1 – Classe Noticia definida com a palavra-chave `final`:

```
<?php

# noticia_final.class.php

final class Noticia
{
      protected $titulo;
      protected $texto;

      function setTitulo($valor)
      {
            $this->titulo = $valor;
      }

      function setTexto($valor)
      {
            $this->texto = $valor;
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

?>
```

Exemplo 1.6.2 – Classe NoticiaPrincipal herda características da classe Noticia:

```
<?php

# noticia_final.php

include_once('noticia_final.class.php');

class NoticiaPrincipal extends Noticia
{
      private $imagem;

      function setImagem($valor)
      {
            $this->imagem = $valor;
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "<img src=\"". $this->imagem ."\"><p>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$titulo = 'Vestibular da Unicamp termina nesta quarta-feira';
$texto  = 'Um dos maiores vestibulares do país acaba nesta quarta-feira,';
$texto .= 'com número recorde de inscritos';
$imagem = 'img_unicamp.jpg';

$not = new NoticiaPrincipal;
$not->setTitulo($titulo);
$not->setTexto($texto);
$not->setImagem($imagem);
$not->exibeNoticia();

echo "<pre>";
print_r($not);
echo "</pre>";

?>
```

O teste realizado com essas classes mostra o seguinte resultado:

Fatal error: Class NoticiaPrincipal may not inherit from final class (Noticia)

1.7- Métodos e propriedades estáticas

Quando definimos métodos ou propriedades como estáticos (utilizando a palavra-chave *static*), estamos permitindo que estes possam ser chamados externamente sem haver a necessidade de estarem no contexto de um objeto, isto é, não é necessário instanciar um objeto para poder acessá-los. Para ter acesso a uma propriedade estática dentro do corpo da classe temos que usar a palavra *self* acompanhada de **::**.

Exemplo 1.7.1 – Propriedade `$nome_jornal` definida como estática na Classe Noticia:

```
<?php

# noticia_estatica.class.php

class Noticia
{
      public static $nome_jornal = 'The Unicamp Post';
      protected $titulo;
      protected $texto;

      function setTitulo($valor)
      {
            $this->titulo = $valor;
      }

      function setTexto($valor)
      {
            $this->texto = $valor;
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "Nome do Jornal: <b>" . self::$nome_jornal . "</b><p>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$titulo = 'Vestibular da Unicamp';
$texto  = 'Um dos maiores vestibulares do país tem número recorde de inscritos';
$not = new Noticia;
$not->setTitulo($titulo);
$not->setTexto($texto);
$not->exibeNoticia();

echo "<p>" . Noticia::$nome_jornal;

?>
```

Quando utilizamos o modificador static em atributos, ao invés de serem alocados n atributos para n objetos, é alocado apenas 1 atributo para n objetos, onde todos os objetos têm acesso ao mesmo atributo.

Dentro da classe filha NoticiaPrincipal, a chamada à métodos ou propriedades estáticas da classe pai ficaria da seguinte forma:

Exemplo 1.7.2 – Propriedade `$nome_jornal` sendo chamada pela sub-classe NoticiaPrincipal:

```
<?php

# noticia_estatica.php

include_once('noticia_estatica.class.php');

class NoticiaPrincipal extends Noticia
{
      private $imagem;

      function setImagem($valor)
      {
            $this->imagem = $valor;
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "Nome do Jornal: <b>" . parent::$nome_jornal . "</b><p>";

            echo "<img src=\"". $this->imagem ."\"><p>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$titulo = 'Vestibular da Unicamp';
$texto  = 'Um dos maiores vestibulares do país tem número recorde de inscritos';
$imagem = 'img_unicamp.jpg';

$not = new NoticiaPrincipal;
$not->setTitulo($titulo);
$not->setTexto($texto);
$not->setImagem($imagem);
$not->exibeNoticia();

?>
```

- Métodos mágicos

Métodos mágicos são métodos com funcionalidades específicas e que podem ser utilizados de acordo com as nossas necessidades. Continuaremos a utilizar o nosso exemplo das classes de notícias.

- **__set** : Este método pode ser declarado em qualquer classe e será executado toda vez que for atribuído algum valor à alguma propriedade do objeto. Ou seja, ele intercepta a atribuição de valores à propriedades de um objeto. Porém, para que este método funcione, estas propriedades devem estar definidas como `protected` ou `private`. Digamos que o título e o texto das nossas notícias devam seguir um tamanho pré-definido. Como fazer esta verificação e atribuir o valor correto nas propriedades automaticamente?

Exemplo 1.8.1 – Verificando a validando o tamanho dos campos das propriedades `$titulo` e `$texto` da classe Noticia:

```
<?php

# noticia_metodo_magico_1.php

class Noticia
{
      protected $titulo;
      public $texto;

      function __set($propriedade, $valor)
      {
            if ( ($propriedade == 'titulo') && (strlen($valor) > 40) )
            {
                  echo "A propriedade <b>$propriedade</b> deve conter
                        no máximo 40 caracteres<p>";
            }

            if ( ($propriedade == 'texto' && strlen($valor) > 100) )
            {
                  echo "A propriedade <b>$propriedade</b> deve conter
                        no máximo 100 caracteres<p>";
            }
      }

      function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$titulo = 'Vestibular da Unicamp termina nesta quarta-feira';
$texto  = 'Um dos maiores vestibulares do país acaba nesta quarta-feira...';

$c_tit = strlen($titulo);
echo "Titulo: ".$c_tit."<p>";
$c_txt = strlen($texto);
echo "Texto: ".$c_txt."<p>";

$not = new Noticia;
$not->titulo = $titulo;
$not->texto = $texto;
$not->exibeNoticia();

echo "<pre>";
print_r($not);
echo "</pre>";

?>
```

- **__get** : Este método pode ser declarado em qualquer classe e será executado toda vez que for solicitado o retorno do valor de alguma propriedade de um objeto. Como em `__set()`, este método funciona apenas com as propriedades que estiverem definidas como `protected` ou `private`.

Exemplo 1.8.2 – Retornando o valor da propriedade `$titulo` com o método `__get()`:

```
<?php

# noticia_metodo_magico_2.php

class Noticia
{
      protected $titulo;

      function __get($propriedade)
      {
            if ( ($propriedade == 'titulo') )
            {
                  echo "Retornando o valor da propriedade <b>$propriedade</b>!";
                  return $this->titulo;
            }
      }

      function setTitulo($valor)
      {
            $this->titulo = $valor;
      }
}

$titulo = 'Vestibular da Unicamp termina nesta quarta-feira';

$not = new Noticia;
$not->setTitulo($titulo);
echo "<p>Título: " . $not->titulo . "</p>";

?>
```

- **__autoload** : Ao criarem aplicações orientadas à objeto, os desenvolvedores colocam a definição de cada classe em um arquivo PHP. Um dos maiores contratempos é ter de escrever uma longa lista de `includes` no início de cada script (uma chamada para cada classe necessária). No PHP5 é possível definir uma função `__autoload()`, que é automaticamente chamada quando se tenta usar uma classe que ainda não foi definida. Ao chamar essa função o "scripting engine" tem uma chance para carregar a classe antes que o PHP falhe com erro. Os próximos dois scripts exemplificam este recurso:

Exemplo 1.8.3 – Script que contém a definição da Classe Noticia :

```
<?php

# noticia_metodo_magico_3.php

class Noticia
{
      public $titulo;
      public $texto;

      function setTitulo($valor)
      {
            $this->titulo = $valor;
      }

      function setTexto($valor)
      {
            $this->texto = $valor;
      }


      function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

?>
```

Exemplo 1.8.4 – Script que utiliza a Classe Noticia, chamando-a através de `__autoload()`:

```
<?php

# noticia_metodo_magico_4.php

function __autoload($classe)
{
      if ($classe == 'Noticia')
      {
            echo "Chamado a Classe <b>$classe</b>";
            include_once('noticia_metodo_magico_3.php');
      }
}

class NoticiaPrincipal extends Noticia
{
      public $imagem;

      function setImagem($valor)
      {
            $this->imagem = $valor;
      }


      function exibeNoticia()
      {
            echo "<center>";
            echo "<u><img src=\"". $this->imagem ."\"></u><p>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }
}

$not = new NoticiaPrincipal;

$not->titulo = 'Vestibular da Unicamp termina nesta quarta-feira';
$not->texto  = 'Um dos maiores vestibulares do país acaba nesta quarta-feira,';
$not->texto .= 'com número recorde de inscritos';
$not->imagem = 'img_unicamp.jpg';
$not->exibeNoticia();

?>
```

- ✔ **__clone** : Criar uma cópia de um objeto com propriedades totalmente replicadas nem sempre é o comportamento desejado. Caso a instância de um objeto seja apenas atribuída à uma variável (Exemplo 1.8.5), o objeto não será clonado, será apenas criada uma nova referência a esse objeto (Exemplo 1.8.6):

Exemplo 1.8.5 – Criação de uma referência para um objeto:

```
$not = new Noticia;
$not_2 = $not;
```

Exemplo 1.8.6 – Exemplo com a Classe Noticia:

```
<?php

# noticia_metodo_magico_5.php

class Noticia
{
	public $titulo;
	public $texto;

	function exibeNoticia()
	{
		echo "<center>";
		echo "<b>". $this->titulo ."</b><p>";
		echo  $this->texto;
		echo "</center><p>";
	}
}

$not = new Noticia;
$not->titulo = "Unicamp 40 anos";
$not->texto = "No ano de 2006 a Unicamp completou 40 anos de história!";
$not->exibeNoticia();

echo "<center><b>==============================================</b></center>";
$not2 = $not;  // Criada a referência para o objeto contido em $not
$not2->titulo = "Economia debate finanças mundiais e estratégias";
$not2->texto = "Começa hoje, no auditório do Instituto de Economia da Unicamp..";
$not2->exibeNoticia();
echo "<center><b>==============================================</b></center>";

$not->exibeNoticia();

echo "<pre>";
print_r($not);
echo "</pre><p>";

echo "<pre>";
print_r($not2);
echo "</pre><p>";

?>
```

Para que o objeto seja realmente clonado, é necessário utilizar a palavra-chave `clone`. E caso o método `__clone()` esteja definido na classe deste objeto, ele será executado durante esta clonagem:

Exemplo 1.8.7 – Executando o método mágico `__clone()` de Noticia:

```
<?php

# noticia_metodo_magico_6.php

class Noticia
{
      public $titulo;
      public $texto;

      function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }

      function __clone()
      {
            echo "<p>Obj. com o título <b>" . $this->titulo . "</b> Clonado</p>";
      }
}

$not = new Noticia;
$not->titulo = "Unicamp 40 anos";
$not->texto = "No ano de 2006 a Unicamp completou 40 anos de história!";
$not->exibeNoticia();

echo "<center><b>================================================</b></center>";
$not2 = clone $not;
$not2->titulo = "Economia debate finanças mundiais e estratégias";
$not2->texto = "Começa hoje, no auditório do Instituto de Economia da Unicamp..";
$not2->exibeNoticia();
echo "<center><b>================================================</b></center>";

$not->exibeNoticia();

echo "<pre>";
print_r($not);
echo "</pre><p>";

echo "<pre>";
print_r($not2);
echo "</pre><p>";

?>
```

- **__toString** : O método `__toString()` é chamado toda vez que se invoca a função *print* ou *echo* para um objeto. Nesses casos o php retornará um erro. É aí que entra o método __toString, que pode imprimir informações relevantes ao objeto. O print_r($objeto) também pode ser utilizado.

  Esse método permite que uma classe decida como se comportar quando convertida para uma string:

Exemplo 1.8.8 – Utilizando o método `__toString()`:

```
<?php

# noticia_metodo_magico_7.php

class Noticia
{
      public $titulo;
      public $texto;

      function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }

      function __toString()
      {
            return "<p>Classe <b>Noticia</b></p>";
      }
}

$not = new Noticia;
echo $not;

?>
```

- **__call** : Esse método será chamado toda vez que for solicitada a execução de algum método inexistente em determinada classe:

Exemplo 1.8.9 – Utilizando o método __call():

```
<?php

# noticia_metodo_magico_8.php

class Noticia
{
      public $titulo;

      function exibeNoticia()
      {
            echo "<center>";
            echo "<b>". $this->titulo ."</b><p>";
            echo  $this->texto;
            echo "</center><p>";
      }

      function __call($metodo, $arg)
      {
            $this->titulo = $arg[0];
            $this->texto = $arg[1];
            echo "Método Chamado: <b>$metodo</b><br>";
            echo "Adicionado a <b>Titulo</b>, o valor " . $arg[0] . "<br>";
            echo "Adicionado a <b>Texto</b>, o valor " . $arg[1];
      }
}

$not = new Noticia;
$not->setTituloTexto('Titulo Adicionado','Texto Adicionado');

echo "<pre>";
print_r($not);
echo "</pre><p>";

?>
```

Capítulo 2

Manipulação de arquivos texto

Esse capítulo aborda de uma maneira fácil e objetiva a manipulação de arquivos texto através do PHP.

Abertura de arquivo: função fopen

Essa função retorna um identificador do arquivo. Esse identificador deve ser armazenado em uma variável, pois será utilizado posteriormente nas funções de leitura, gravação e fechamento do arquivo.

```
$arquivo = "teste.txt";
$abertura = fopen($arquivo, "w");
```

Os modos de abertura são:

- **r** : Somente leitura.
- **r+** : Leitura e gravação. Se o arquivo já existir, irá gravar no início do arquivo.
- **w** : Somente gravação. Se o arquivo já existir, irá apagar todo o conteúdo prévio.
- **w+** : Gravação e Leitura. Se o arquivo já existir, irá apagar todo o conteúdo prévio. Caso o arquivo não exista, ele será criado.
- **a** : Somente gravação. Caso o arquivo exista irá gravar no final do arquivo. Caso o arquivo não exista, ele será criado.
- **a+** : Gravação e leitura. Caso o arquivo exista irá gravar no final do arquivo. Caso o arquivo não exista, ele será criado.

Gravação de arquivo: função fwrite

Essa função recebe como argumentos o identificador de arquivo (obtido no processo de abertura) e uma string (conteúdo do arquivo). Retorna o número de caracteres gravados.

```
$conteudo = "Isto é um teste";
$gravacao = fwrite($abertura, $conteudo);
```

Leitura de arquivo: função fread

Essa função recebe como argumentos o identificador do arquivo (obtido no processo de abertura) e o tamanho do arquivo em bytes como segundo

argumento. Retorna o conteúdo do arquivo, que deve ser armazenado em uma variável.
Para obter o tamanho do arquivo, pode-se utilizar a função *filesize()* ou então o resultado que retornou da função de gravação, uma vez que a função *fwrite()*, como já vimos, retorna o número de caracteres gravados.

```
$abertura = fopen($arquivo, "r");
$leitura = fread($abertura,filesize($arquivo));
```

Reposicionando o ponteiro num determinado registro do arquivo: função fseek

Essa função recebe como argumentos o identificador de arquivo (obtido no processo de abertura) e a posição do arquivo em que deve começar a fazer a leitura (no exemplo abaixo, a posição 0 significa reposicionar o ponteiro no início do arquivo).

```
fseek($abertura, 0);
```

Fechamento de arquivo: função fclose

Para finalizar o uso de um arquivo, deve-se fechá-lo, assim todas as alterações serão salvas. Não é necessário utilizar nenhum argumento além do próprio identificador do arquivo.

```
fclose ($abertura);
```

No exemplo a seguir, reunimos essas cinco funções num único script:

```
<?php
      # teste_arquivo.php
      $arquivo = "teste.txt";
      $conteudo="Isto é um teste";
      $abertura=fopen("$arquivo","w+");
      $gravacao = fwrite($abertura, $conteudo);
      echo "Número de caracteres gravados: $gravacao";
      # Reposiciona o ponteiro no início do arquivo
      fseek($abertura, 0);
      $leitura = fread($abertura, filesize($arquivo));
      fclose ($abertura);
      echo "<br> Conteúdo do arquivo: $leitura";
?>
```

Exercício

Reescreva o código acima, seguindo os conceitos da programação orientada a objetos. Utilize a classe “arquivo”, disponibilizada a seguir.

A resposta desse exercício está na página 50.

```
<?php
# arquivo.class.php

class arquivo
{
      # Atributos
      protected $abertura;
      protected $gravacao;
      protected $leitura;

      # Métodos
      # Abertura do arquivo, recebe como parametros o nome do arquivo
      # e o tipo de abertura
      function abreArq ($nome_arq, $tipo_abertura)
      {
            $this->abertura = fopen("$nome_arq", "$tipo_abertura");
      }

      # Gravação do arquivo, a função fwrite recebe como parâmetro o conteúdo
      # e o identificador de abertura do arquivo
      function gravaArq ($conteudo)
      {
            $this->gravacao = fwrite($this->abertura, $conteudo);
      }

      # Posicionamento do ponteiro numa posição específica do arquivo
      function posicionaArq ()
      {
        fseek($this->abertura, 0);
      }

      # Leitura do arquivo, a função fread utiliza como parâmetro
      # o identificador de abertura do arquivo e o tamanho do arquivo
      function leArq ()
      {
            $this->leitura = fread($this->abertura, $this->gravacao);
      }

      # Fechamento do arquivo, a função fclose utiliza como parâmetro
      # o identificador de abertura do arquivo
      function fechaArq ()
      {
            fclose ($this->abertura);
      }

      # Exibe o conteúdo do arquivo, utiliza o resultado obtido
      # pelos métodos de gravação e leitura
      function exibeArq ()
      {
        echo "Número de caracteres gravados: $this->gravacao";
            echo "<br>Conteúdo do arquivo: $this->leitura";
      }
}

?>
```

Outras funções úteis para a manipulação de arquivos texto:

função feof

Testa pelo fim de arquivo (eof) em um ponteiro de arquivo.

função fgets

Lê uma linha de um ponteiro de arquivo e retorna uma string. A leitura termina quando encontra fim de linha ou fim de arquivo.

A seguir temos um exemplo que utiliza essas duas funções:

```
<?php

      $abertura = fopen("teste.txt","r");
      // Testa pelo fim de arquivo (eof)
      while (!feof($abertura)) {
             // fgets lê uma linha de um ponteiro de arq
             $buffer = fgets($abertura);
             echo $buffer . "<Br>";
      }
      fclose($abertura);
?>
```

Capítulo 3

Extensões do PHP5 para a manipulação de documentos XML

Ensina a manipular arquivos XML utilizando as extensões do PHP.

Além de funções específicas para a manipulação de arquivos XML, o PHP5 trabalha com 3 extensões XML, cada qual com suas características específicas:

DOM (Document Object Model): é a extensão com mais recursos, porém exige muito trabalho, mesmo para o XML mais simples. Além disso utiliza mais memória.

SAX (Simple API for XML): extensão original XML do PHP. Utiliza menos memória que a DOM, porém, necessita frequentemente de código PHP complexo.

SimpleXML: essa extensão já vem habilitada por padrão no PHP5. Com ela, trabalha-se com documentos XML como se fossem objetos nativos da linguagem PHP. É mais simples de se usar do que as extensões mencionadas anteriormente.

Agora vamos saber mais sobre os tipos de arquivos com os quais vamos trabalhar.

O que é XML

XML (eXtensible Markup Language) é uma linguagem de marcação que permite a troca de informações de forma estruturada atráves da Internet. É composta por tags (assim como o HTML), só que, enquanto o HTML trabalha com tags pré-definidas, o XML permite que o desenvolvedor crie as suas próprias tags definindo assim a sua própria linguagem de marcação. Um arquivo xml nada mais é do que um arquivo texto com marcações, que permitem um maior controle sobre as informações contidas nesse arquivo e que facilitam a troca de dados entre programas escritos em linguagens diferentes.

XML formato RSS

Ainda tendo como tema um site de notícias, vamos demonstrar como criar um arquivo XML no formato RSS e como utilizar a extensão SimpleXML para ler esse arquivo.
RSS é um formato de distribuição de informações pela Internet que permite ao usuário ter acesso a notícias de várias fontes simultaneamente, sem ter que navegar pelos respectivos sites. Esses arquivos são escritos em XML e também são conhecidos como “feeds”.

Por meio de programas leitores de RSS, o usuário cadastra o endereço do feed desejado e passa a receber as notícias em seu computador em tempo real, num procedimento semelhante ao utilizado pelos gerenciadores de e-mail.

Os arquivos RSS trazem um conjunto de tags específicas. As principais são: *rss*, *channel* e *item* .
A tag *rss* é a raíz do documento XML, contendo todas as demais. Dentro da tag *rss* temos apenas uma tag *channel*. Na tag *channel* temos várias outras tags que trazem informações sobre o documento rss: Quem o gerou, o que contém, data da geração, etc. Dentro da tag *channel* teremos também várias tags *item*. Cada tag *item* representa uma notícia, e é composta por outras tags que descrevem a notícia, tais como *título*, *link* e *descrição*.

A seguir, temos a estrutura básica de um arquivo RSS:

```
<rss>
	<channel>
		<item>
			<title></title>
			<link></link>
			<description></description>
		</item>
		<item>
			<title></title>
			<link></link>
			<description></description>
		</item>
	</channel>
</rss>
```

Exercício:

Vamos criar um *feed* utilizando PHP e MySQL. Inicialmente, incluiremos dados numa tabela de notícias através de um formulário. Em seguida, um programa acessará esses dados e gerará um arquivo XML no formato RSS. Para finalizar, outro programa vai ler o feed e exibir as notícias.
Nesse exercício, utilizaremos os conceitos da Programação Orientada a Objetos, vistos anteriormente.

Passo 1: Definição da tabela “noticia” - base de dados: “capitulo3”.
Utilize a ferramenta PhPMyAdmin.

Campos da tabela:

id (int, 3, chave, auto-increment)
titulo (varchar, 100)
link (varchar, 100)
descricao (text)
data_publ (date)

Passo 2: Definição das classes
Para esse exercício criaremos duas classes, que ficarão armazenadas na pasta *classes*:
Classe db: utilizada nos acessos ao banco de dados.
Classe feed: utilizada na manipulação dos dados do arquivo RSS.

A seguir, temos a definição dessas classes:

db.class.php :

```
<?php
# db.class.php

class db
{
	public $conexao;

	function __construct($dominio, $usuario, $senha, $db)
	{
		$this->conexao = mysql_connect($dominio, $usuario, $senha);
		mysql_select_db($db, $this->conexao);
	}

	function DBError()
	{
		echo mysql_error($this->conexao);
	}

	function insert($tabela, $campos)
	{
		$declar = "INSERT into $tabela values $campos";
		return mysql_query ($declar);
	}

	function select($tabela, $campos, $condicao)
	{
		$declar = "SELECT $campos from $tabela $condicao";
		return mysql_query ($declar);
	}

	function delete($tabela, $condicao)
	{
		$declar = "DELETE from $tabela $condicao";
		return mysql_query ($declar);
	}

	function update($tabela, $campos, $condicao)
	{
		$declar = "UPDATE $tabela SET $campos WHERE $condicao";
		return mysql_query ($declar);
	}
}

?>
```

A classe **db** possui um método construtor que fará a conexão com o banco de dados assim que essa classe for instanciada. Também disponibiliza métodos que fazem a inclusão, seleção, exclusão e alteração em tabelas.

feed.class.php :

```
<?php

class feed
{
	public $obj_feed;

	function __construct($nome_feed)
	{
		$this->obj_feed = simplexml_load_file($nome_feed);
	}

	function exibeFeed()
	{
		foreach ( $this->obj_feed->channel->item as $noticia )
		{
			echo '<br>';
			echo $noticia->title . '<br>';
			echo "<a href=\"$noticia->link\">$noticia->link</a>".'<br>';
			echo $noticia->description . '<br>';
		}
	}
}

?>
```

No método construtor da classe feed utilizamos a função *simplexml_load_file*, que lê o conteúdo de um arquivo XML e retorna um objeto.
No método exibeFeed utilizamos o comando *foreach*, que oferece uma maneira fácil de trabalhar com matrizes e funciona somente com arrays. O *foreach* varre uma dada matriz e em cada 'loop', o valor do elemento corrente é atribuído a uma variável (*$noticia*) e o ponteiro interno da matriz é avançado em uma posição.

Mais informações sobre essa função na página 48.

Passo 3: Formulário para a entrada de dados (form_noticia.html)

```
<html>
<head>
<title>Formulário</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<p><font size="2" face="Verdana, Arial, Helvetica, sans-serif"><strong>Inserir
Notícias </strong></font></p>
<form name="form1" method="post" action="inclui_noticia.php">
 <table width="50%" border="0" cellspacing="10" cellpadding="0">
 <tr>
 <td width="20%"><strong><font size="2" face="Verdana, Arial, Helvetica, sans-
serif">Título:</font></strong></td>
 <td width="80%"><input name="titulo" type="text" id="titulo" size="50"
maxlength="100"></td>
 </tr>
 <tr>
 <td width="20%"><strong><font size="2" face="Verdana, Arial, Helvetica, sans-
serif">Link:</font></strong></td>
 <td width="80%"><input name="link" type="text" id="link" size="50"
maxlength="100"></td>
 </tr>
 <tr>
 <td width="20%"><strong><font size="2" face="Verdana, Arial, Helvetica, sans-
serif">Descrição:</font></strong></td>
 <td width="80%"><textarea name="descricao" cols="50" rows="4"
id="descricao"></textarea></td>
 </tr>
 </table>
 <p>
 <input type="submit" name="Submit" value="Enviar">
 </p>
 </form>
</body>
</html>
```

Passo 4: O script inclui_noticia.php fará a inclusão dos dados recebidos pelo formulário na tabela noticia, utilizando a classe DB.

```
<?php
      # inclui_noticia.php
      include_once ('classes/db.class.php');

      // Recebe dados do formulário
      $titulo = $_POST["titulo"];
      $link = $_POST["link"];
      $descricao = $_POST["descricao"];

      // obtem data de publicação
      $data_publ = date("Y").'-'.date("m").'-'.date("d");

      // Instancia um objeto da classe db
      $inc_not = new db('localhost', 'root', 'vertrigo', 'capitulo3');
      $inc_not->DBError();
      $campos = "(' ', '$titulo', '$link', '$descricao', '$data_publ')";
      $result = $inc_not->insert('noticia', $campos);

      If ($result) {
            echo "Inclusao OK <br><br> <a href=\"form_noticia.html\">Voltar</a>";
      }
      else {
            echo "Erro na Inclusao";
      }
?>
```

Passo 5: Script que gera o arquivo XML (Feed) a partir dos dados da tabela noticia (gera_feed.php). Nesse script utilizaremos as classes “DB” e “Arquivo”.

```
<?php

# gera_feed.php

function __autoload($classe)
{
	include_once ("classes/$classe.class.php");
}

// Obtem data atual
$data_atual = date("Y").'-'.date("m").'-'.date("d");

// Instancia um objeto da classe db
$sel_not = new db('localhost', 'root', 'vertrigo', 'capitulo3');
$sel_not->DBError();
$condicao = "where data_publ = '$data_atual'";
$result = $sel_not->select('noticia', '*', $condicao);

// Verifica se encontrou algum registro
$row = mysql_num_rows($result);
if ($row > 0)
{
	// Determina o nome do arquivo XML que será criado
	$arquivo = "feed.xml";

	// Instancia um objeto da classe arquivo e abre o arquivo
	$arquivo_xml = new arquivo();
	$arquivo_xml->abreArq($arquivo,'w');

	$conteudo = "<?xml version='1.0' encoding='ISO-8859-1'?>";
	$conteudo .= "<rss version='2.0'>";
	$conteudo .= '<channel>';
	$conteudo .= '<title>Seu Site</title>';
	$conteudo .= '<link>http://www.seusite.com.br</link>';
	$conteudo .= '<description>Descrição de seu site</description>';
	$conteudo .= '<language>pt-br</language>';
	$conteudo .= '<copyright>Copyright de seu site</copyright>';
	$conteudo .= '<webmaster>webmaster@seusite.com.br</webmaster>';

	while ($result2 = mysql_fetch_assoc($result)) {

		// Monta as tags referentes as noticias
		$conteudo .= "<item>";
		$conteudo .= "<title>$result2[titulo]</title>";
		$conteudo .= "<link>$result2[link]</link>";
		$conteudo .= "<description>$result2[descricao]</description>";
		$conteudo .= "</item>";

	}

	//Fecha as tags channel e rss
	$conteudo .= '</channel>';
	$conteudo .= '</rss>';

	$arquivo_xml->gravaArq($conteudo);
	$arquivo_xml->fechaArq();

	// Mensagem
	echo "O arquivo <b>".$arquivo."</b> foi gerado com sucesso !";
}

?>
```

O script gera_feed.php irá gerar o arquivo “feed.xml” com formato RSS.

O conteúdo desse arquivo ficará parecido com o código a seguir:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rss version="2.0">
<channel>
<title>Seu Site</title>
<link>http://www.seusite.com.br</link>
<description>Descrição de seu site</description>
<language>pt-br</language>
<copyright>Copyright de seu site</copyright>
<webmaster>webmaster@seusite.com.br</webmaster>
<item>
<title>Noticia 1</title>
<link>URL da Notícia 1</link>
<description>Descrição da Notícia 1 </description>
</item>
<item>
<title>Noticia 2</title>
<link> URL da Notícia 2</link>
<description> Descrição da Notícia 2 </description>
</item>
<item>
<title>Noticia n</title>
<link> URL da Notícia n </link>
<description> Descrição da Notícia n </description>
</item>
</channel>
</rss>
```

Passo 6 – exercício: Crie um script (exibe_noticia.php) para ler o feed criado e exibir o seu conteúdo. Utilize a classe feed.

A resposta desse exercício está na página 50.

Obs: Após testar com sucesso o script **exibe_noticia.php**, passando como parâmetro o arquivo *feed.xml,* faça um novo teste, dessa vez passando como parâmetro o feed da Folha Online Ilustrada.
Url do feed: **http://feeds.folha.uol.com.br/folha/ilustrada/rss091.xml**

Outros recursos da extensão SimpleXML

O SimpleXML também disponibiliza outros recursos bastante úteis, como a função *simplexml_load_string*, utilizada para ler uma string XML, e a função *xpath*, que permite acessar uma informação específica dentro de um objeto.

A seguir, um exemplo que utiliza essas duas funções. Primeiramente, transformamos o conteúdo do nosso arquivo *feed.xml* em uma string, com o auxílio da função *implode*. Em seguida lemos essa string com a função *simplexml_load_string* e obtemos o título da primeira notícia usando *xpath*:

```
<?php

      $string = file("feed.xml");
      $string = implode(" ", $string);
      $xml = simplexml_load_string($string);

      /* Procurando pelo título da primeira notícia */
      $result = $xml->xpath('/rss/channel/item/title');

      echo $result[0];

?>
```

No código acima o *xpath* vai retornar um array para a variável $result, contendo todos os elementos *title* (títulos) do arquivo. Ao indexarmos a variável, estamos escolhendo a posição do array que queremos exibir.

Obs: Também podemos ler o nosso arquivo RSS com um dos muitos leitores de RSS disponíveis, é só cadastrar o endereço do feed desejado :

Feedreader: http://www.feedreader.com/ (Windows) Leitor de notícias do Thunderbird: http://www.mozilla.com/thunderbird/ (Windows/Linux).

Sites agregadores de conteúdo como o Netvibes – http://www.netvibes.com/ - também disponibilizam uma interface para a inclusão e leitura de feeds.

Capítulo 4

Templates com POO

Introdução

Em muitos casos, o código de uma aplicação para a Web é desorganizado, com muitos códigos HTML e instruções PHP escritas de qualquer jeito. Aliando o conceito de Programação Orientada à Objetos a um bom planejamento e organização do seu template, é possível deixar o código mais enxuto e organizado, facilitando assim a manutenção das aplicações.

Neste capítulo veremos como separar os códigos HTML, CSS e Javascript, do código PHP da aplicação.

Ao separarmos a camada lógica (código PHP), da camada de apresentação (layout do site), possibilitamos que programador e webdesigner trabalhem de forma independente.

1.1– O template

No projeto final utilizaremos um template de um site de notícias. Os scripts referentes a esse projeto serão disponibilizados para fins de estudo. O objetivo é fornecer aos alunos um ponto de partida para que eles exercitem os conceitos apresentados no treinamento e completem o projeto sozinhos.

Dividimos o template em 7 partes:

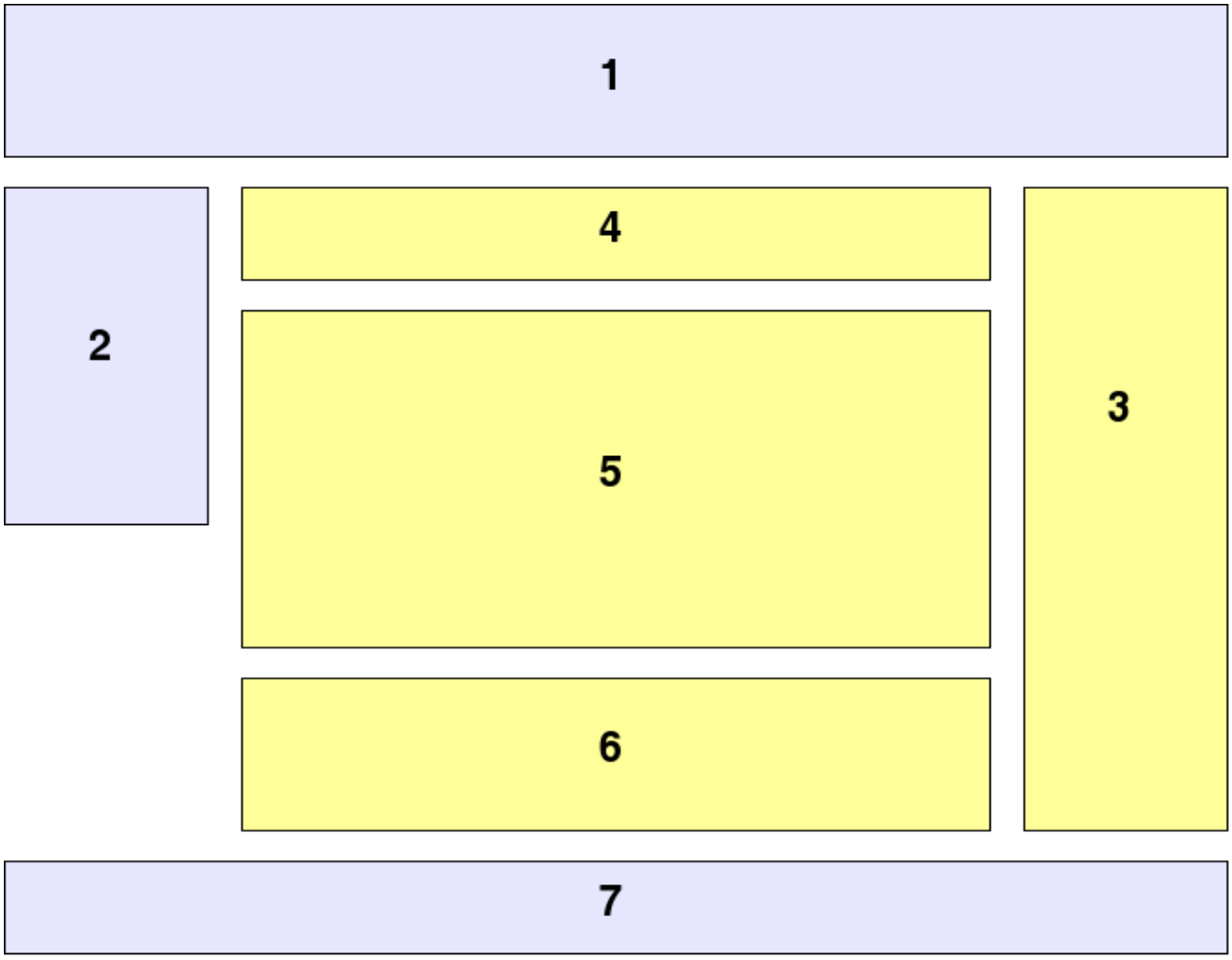

Figura 1.1 – Divisão do template do Projeto Final.

1. **Cabeçalho**: Definições da página e imagens.
2. **Menu** : Na coluna esquerda encontra-se-se o menu de navegação.
3. **Notícia de Eventos**: Na coluna direita encontram-se as notícias referentes à eventos importantes.
4. **Notícia Principal** : Nesta seção encontra-se a notícia principal, a manchete do site.
5. **Notícia do Dia** : Nesta seção encontram-se as notícias do dia que não são manchetes.
6. **Notícia de Última Hora** : Nesta seção encontram-se as notícias de última hora.
7. **Rodapé**: informações sobre os responsáveis pelo site e a política de privacidade da página.

1.1.2 – Classes

- **Noticia.class.php :** Este script contém a classe Noticia, utilizada na maioria dos scripts para manipulação das informações da página principal.

- **template.class.php :** Contém as classes responsáveis pela manipulação do template da página. Tudo que está relacionado com a apresentação da página encontra-se neste script.

- **dbconnect.php :** Contém a classe responsável pela manipulação das informações armazenadas no banco de dados MySQL. Qualquer operação com o banco de dados passa por objetos instanciados desta classe.

- **index.php :** Este script organiza e instancia objetos que estão definidos nos scritps anteriores, efetivando a apresentação da página principal.

Obs: Os scripts do projeto serão disponibilizados como material complementar.

1.1.3 – Funções

A seguir, apresentamos algumas funções do PHP que serão utilizadas no projeto:

Include_once()

Inclui um script em php na página, apenas uma vez, ou seja, quando utilizamos várias vezes um mesmo script na página e se por acaso dentro deste script eu fazer o uso de outro que já foi usado na hierarquia, o include once irá importá-lo uma única vez. Já quando se utiliza o include normal, o script será importado para cada chamada.

foreach()

```
foreach ($array as $valor) {
  comandos
 }
foreach ($array as $chave => $valor) {
  comandos
 }
```

O foreach é uma maneira fácil de se mover dentro de um array. O que ele faz é repetir comandos enquanto existirem elementos dentro de um array. Existem duas sintaxes para usar o foreach, cada uma delas gerando um resultado diferente.

```
<?php
    $frutas = array('banana','maçã','mamão','manga');

    // O fluxo de execução vai executar/repetir os comandos ligados ao
    // foreach até acabarem os valores do array $frutas. A cada
    // repetição o valor atual é atribuído a variável $valor

    foreach($frutas as $valor) {
      echo $valor; // É exibida um dos valores de $frutas a cada repetição
    }


    // Essa sintaxe do foreach faz a mesma coisa que a sintaxe acima,
    // porém ela atribui o valor da chave atual à variável $chave
    // (além do valor atual à variável $valor).

    foreach($frutas as $chave => $valor) {
      echo $chave . '-' . $valor;
      // retorna algo como '2 - mamão' a cada repetição
    }
?>
```

array_push()

Trata *array* como uma pilha, e adiciona as variáveis passadas como argumentos no final de *array*. O comprimento de *array* aumenta de acordo com o número de variáveis adicionadas.

Exemplo de **array_push()**

```
<?php
  $cesta = array("laranja", "morango");
  array_push($cesta, "melancia", "batata");
  print_r($cesta);
?>
```

Neste caso `$stack` teria os seguintes elementos:

```
Array
(
    [0] => laranja
    [1] => morango
    [2] => melancia
    [3] => batata
)
```

Respostas dos exercícios:

Exercício do capítulo 2 – pág. 32

Gravação e leitura de um arquivo texto utilizando a classe “Arquivo”.

```
<?php
# teste_oo_arquivo.php
# objetivo: gravar conteúdo em um arquivo texto e exibir o conteúdo gravado

// Inclui o script onde foi definida a classe a ser utilizada
include_once ('classes/arquivo.class.php');

# Inicializa variáveis que serão passadas como parâmetros
$arquivo = "teste.txt";
$conteudo = "Isto é um teste";

# Instancia um objeto da classe arquivo
$arq = new arquivo();

# Chama o método de abertura de arquivo no modo escrita
$arq->abreArq($arquivo,'w+');

# Chama o método de gravação de arquivo
$arq->gravaArq($conteudo);

# Chama o método de abertura de arquivo no modo leitura
$arq->posicionaArq();

# Chama o método de leitura de arquivo
$arq->leArq();

# Chama o método de fechamento de arquivo
$arq->fechaArq();

# Chama o método que exibe o conteúdo do arquivo
$arq->exibeArq();

?>
```

Exercício do capítulo 3 – pág. 43

Passo 6 - script exibe_noticia.php

```
<?php

      # exibe_noticia.php

      include_once ('classes/feed.class.php');

      $noticia = new feed('feed.xml');
      $noticia->ExibeFeed();

?>
```