Autores:
Fabio Santos
Alexandre Arroyo

DTTEC
Centro de Computação
Unicamp

Licenciamento de Uso

Índice

Capítulo 1 – Programação Orientada a Objetos.....	5
Classes e objetos.....	6
Herança	7
Método construtor e destrutor.....	8
Encapsulamento.....	10
Interfaces.....	13
Classes abstratas.....	15
Palavra-chave 'final'.....	17
Métodos e propriedades estáticas.....	19
Métodos mágicos.....	21
Capítulo 2 – Manipulação de arquivos texto.....	31
Capítulo 3 – Simple XML.....	35
Capítulo 4 – Templates com OOP.....	47

Última atualização em 24/05/2006

Capítulo 1

Programação Orientada à Objetos 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.

1. Classes e Objetos

Classe é a estrutura mais fundamental para a criação de um objeto. Uma classe nada mais é que um conjunto organizado de variáveis (propriedades ou atributos) e funções (métodos), que futuramente será utilizado como um novo tipo e instanciará um 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. 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();

?>
```

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, 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 suporte à exibição de imagens nas notícias principais. Nestas 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 têm porquê serem os mesmos que os definidos nas classes “filhas”.

1.2– Método Construtor e Destrutor

- Construtor : É um método que contém o nome reservado `__construct()` , e que não precisa ser chamado da forma convencional, pois é executado automaticamente quando instanciamos um objeto. Classes que tem um método construtor chamam-no cada vez que um objeto novo é criado, por isso, é apropriado para qualquer inicialização que o objeto possa vir a precisar antes de ser usado. O método construtor se encarrega de resumir 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 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('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)
      {
            $this->titulo = $valor_tit;
            $this->text   = $valor_text;
            $this->imagem = $valor_img;
      }

      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>";
      }
}

$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. Mesmo assim podemos utilizá-lo, através de uma chamada específica, como no código abaixo:

Exemplo 1.2.3 – Método construtor da Classe Noticia sendo chamado em NoticiaPrincipal:

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

- 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. Como no método construtor, o método destrutor possui um nome reservado, o `__destruct()`. O exemplo abaixo 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 é 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 desta 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 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_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 = 'ComCiência, de roupa nova, discute genética humana';
$texto  = 'A ComCiência, revista eletrônica produzida pelo Laboratório...';

$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 implementar 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 nos exemplos anteriores é simples. No código abaixo, 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 termina nesta quarta-feira';
$texto  = 'Um dos maiores vestibulares do país acaba nesta quarta-feira,';
$texto .= 'com 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 subclasse para conseguir utilizar suas características. Isso não quer dizer que os métodos destas classes também precisam ser abstratos. Isso é opcional, mas propriedades não podem ser definidas como abstratas. Vamos transformar a classe Noticia em uma classe abstrata e depois herdar suas características para a subclasse 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 abstrato quanto de métodos comuns. Os métodos abstratos não devem conter código, apenas definição. Suas ações serão definidas na subclasse que herdará Noticia, no nosso caso 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 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>";

?>
```

1.6- A palavra-chave 'final'

Classes definidas com a palavra-chave `final` não podem ser herdadas, ou seja, não é possível criar subclasses através destas classes. Definindo métodos desta forma, as subclasses que os herdarem não poderão redefini-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 estas classes nos traz 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 (utlizando 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:

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 termina nesta quarta-feira';
$texto  = 'Um dos maiores vestibulares do país acaba nesta quarta-feira...';
$not = new Noticia;
$not->setTitulo($titulo);
$not->setTexto($texto);
$not->exibeNoticia();

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

?>
```

Dentro da classe filha NoticiaPrincipal, a chamada à métodos ou propriedades estáticos 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 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();

?>
```

1.8- 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;
      protected $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 isso não é mais necessário. Você pode definir uma função `__autoload()` que é automaticamente chamada no caso de você tentar 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 completa 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 completa 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()` 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** : Este 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

Este 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 que deve ser armazenado em uma variável para o uso posterior em funções de leitura, gravação e fechamento.

```
$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.
- **a** : Somente gravação. Caso o arquivo exista irá gravar no final do arquivo.
- **a+** : Gravação e leitura. Caso o arquivo exista irá gravar no final do arquivo.

Gravação de arquivo: função fwrite

Essa função aceita argumentos na forma de um identificador de arquivo e uma string. Retorna o número de caracteres gravados.

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

Leitura de arquivo: função fread

Essa função aceita um identificador de arquivo e um tamanho do arquivo em bytes como segundo argumento.

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

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);
```

Agora vamos juntar essas quatro funções num único exemplo:

Exemplo

```
<?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: $gravacao";
      fseek($abertura, 0);
      $leitura = fread($abertura, filesize($arquivo));
      fclose ($abertura);
      echo "<br> Conteúdo do arquivo: $leitura";
?>
```

Exercício

Vamos retrabalhar o exemplo acima, seguindo os conceitos da programação orientada a objetos. Crie um script que faça a gravação e leitura em um arquivo, utilizando a classe “Arquivo” descrita a seguir (resposta na página 49).

```
<?php

# arquivo.class.php

class Arquivo
{
      protected $abertura;
      protected $gravacao;
      protected $leitura;

      function abreArq ($nome_arq, $tipo_abertura)
      {
            $this->abertura = fopen("$nome_arq", "$tipo_abertura");
      }

      function gravaArq ($conteudo)
      {
            $this->gravacao = fwrite($this->abertura, $conteudo);
      }

      function leArq ()
      {
            $this->leitura = fread($this->abertura, $this->gravacao);
      }

      function fechaArq ()
      {

            fclose ($this->abertura);

      }


      function exibeArq ()
      {
            echo "<br> Conteúdo do arquivo: ". $this->leitura;
      }
}
?>
```

Capítulo 3

Extensões do PHP5 para a manipulação de documentos XML

Aprenda a manipular arquivos XML com o PHP através do suporte nativo do PHP, o SimpleXML.

Além de funções específicas para a manipulação de arquivos XML, o PHP5 trabalha com 5 extensões XML, cada qual com suas características específicas:

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

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

XPath: Essa extensão permite que façamos buscas num documento XML da mesma forma como ocorre numa pesquisa num banco de dados.

XSLT (Extensible Stylesheet Language Transformations): É uma linguagem utilizada para transformar documentos XML em outros documentos dando uma "cara nova". Na verdade, é uma folha de estilo, parecido com o CSS utilizado em páginas HTML. XSLT é facilmente compartilhado entre diferentes aplicativos, mas possui uma sintaxe que pode parecer um pouco complicada.

SimpleXML: facilita a manipulação de arquivos XML, convertendo-os em objetos, e já vem habilitada por default no PHP5. Com ela, trabalha-se com documentos XML como se fossem objetos nativos da linguagem PHP, evitando a utilização do padrão DOM, bem mais complexo.

Nesse treinamento utilizaremos SimpleXML, por ser uma das extensões mais fáceis de se 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. Dessa forma, permite a troca de dados entre programas escritos em diferentes linguagens.

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 formato semelhante ao dos gerenciadores de e-mail.

Os arquivos RSS trazem um conjunto de tags específicas, as principais são: *rss*, *channel* e *item* .

A tag *rss* é o root do documento XML, contém 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 conterá algumas tags descrevendo a notícia, tal como *título*, *link* e *descrição*.

A seguir, temos a estrutura básica de um arquivo RSS e suas tags principais:

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

Exercício:

Vamos criar o nosso próprio feed, um arquivo XML (padrão RSS), utilizando PHP e MySQL. Inicialmente, incluiremos dados numa tabela de notícias, através de um formulário. Em seguida, um programa vai ler esses dados e gerar 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: “topico_xml”)

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;
      public $resultado;

      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 insertTab ($tab, $campos)
      {
            $declar = "INSERT into $tab values $campos";
            $this->resultado = mysql_query ($declar);
      }
```

```
      function selectTab ($tab, $campos, $condicao)
      {
            $declar = "SELECT $campos from $tab $condicao";
            $this->resultado = mysql_query ($declar);
      }

      function deleteTab ($tab, $condicao)
      {
            $declar = "DELETE from $tab $condicao";
            $this->resultado = mysql_query ($declar);
      }

      function updateTab ($tab, $campos, $condicao)
      {
            $declar = "UPDATE $tab SET $campos WHERE $condicao";
            $this->resultado = mysql_query ($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.

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 BD
$obj = new DB('localhost', 'root', 'unicamp', 'noticias');
$obj->DBError();
$campos = "(' ', '$titulo', '$link', '$descricao', '$data_publ')";
$obj->insertTab('noticia', $campos);

if ($obj->resultado == 1)
{
      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

include_once ('classes/db.class.php');
include_once ('classes/arquivo.class.php');

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

// Instancia um objeto da classe DB
$obj = new DB('localhost', 'root', 'unicamp', 'noticias');
$obj->DBError();
$condicao = "where data_publ = '$data_atual'";
$obj->selectTab('noticia', '*', $condicao);

// Verifica se encontrou algum registro
$row = mysql_num_rows($obj->resultado);

if ($row > 0)
{
      // Determina o nome do arquivo XML que será criado
      $arquivo = "feed.xml";

      // Instancia um objeto da classe Arquivo
      // 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 ($result = mysql_fetch_assoc($obj->resultado))
      {

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

Capítulo 4

Templates com OOP

Este forma de organizar suas aplicações Web tem por objetivo tornar seus códigos mais enxutos e fáceis de serem compreendidos, auxiliando desta forma, a manutenção destas aplicações. Será abordada a forma mais básica de implementação deste conceito, que servirá de base para estudo de casos mais complexos.

1. Introdução

Em muitos casos, a escrita de código de uma aplicação para a Web é desorganizada, engessada e de difícil compreensão, com muitos códigos HTML e instruções PHP feitas de qualquer jeito. O impacto disso é o árduo trabalho que se tem ao precisar trocar o template da sua aplicação, trocar as chamadas e instruções SQL relacionadas a um banco de dados que será trocado por outro, entre outros inconvenientes. Utilizando o conceito de Programação Orientada à Objeto e um bom planejamento e organização do seu template, é possível minimizar drasticamente este trabalhoso serviço de manutenção de código. Neste capítulo utilizaremos um exemplo prático que será utilizado em nosso Projeto Final. Este exemplo consiste em mostrar de uma maneira bem simplista e básica, como separar os códigos *Client Side* (HTML, CSS e Javascript), do código PHP de sua aplicação.

1.1– O template utilizado

No endereço http://localhost/php/template , encontra-se a página que será utilizada em nosso Projeto Final, que terá como tema um site de notícias. A organização da sua estrutura encontra-se na Figura 1.1. Dividimos o template em 8 partes, das quais 5 têem seu conteúdo armazenado em banco de dados, que será o MySQL.

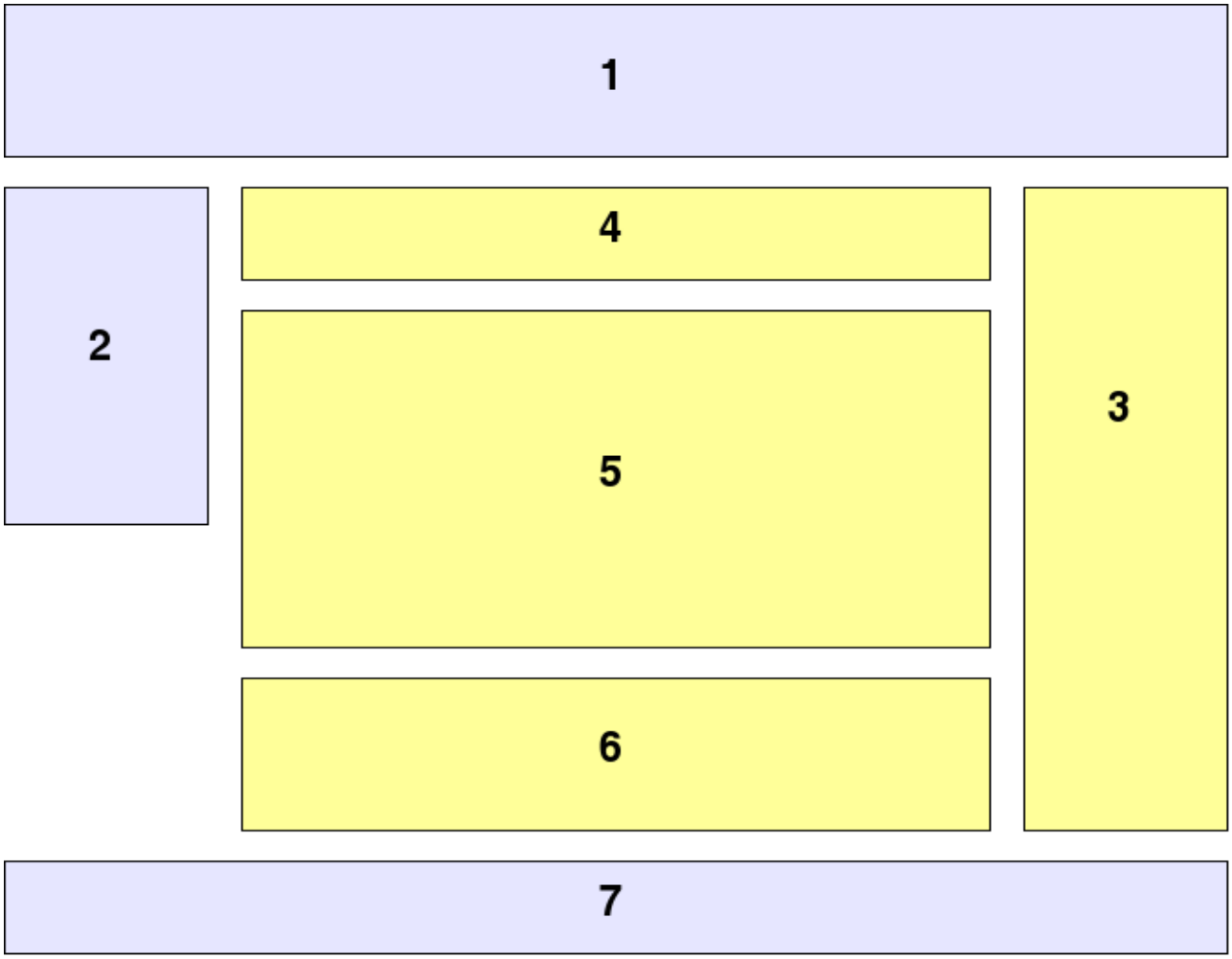
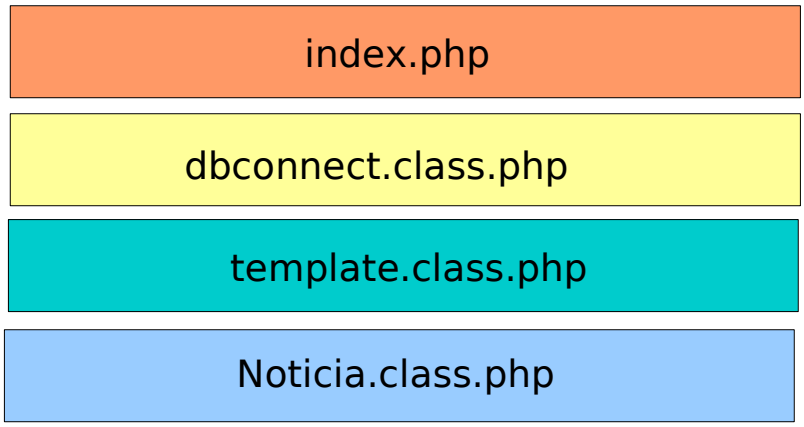


Figura 1.1 – Divisão do template do Projeto Final.

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– Códigos

Quatro scripts são necessários para colocar a página principal em funcionamento. Cada um deles desempenha um papel diferete na construção desta página. A Figura 1.2 ilustra o funcionamento deste scripts:



- **Noticia.php :** Este script contém a classe Noticia, que será 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 que estão armazenadas no banco de dados MySQL. Qualquer operação com o banco de dados passa por objetos instanciados desta classe.

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

Resposta do exercício do tópico 2

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

```
<?php
# teste_oo_arquivo.php
# objetivo: gravar conteúdo num arquivo texto e em seguida exibir esse conteúdo

      // 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 a classe
      $arq = new Arquivo();

      # Chama o método de abertura de arquivo (escrita)
      $arq->abreArq($arquivo,'w');

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

      # Chama o método de abertura de arquivo (leitura)
      $arq->abreArq($arquivo,'r');

      # 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();

?>
```

Resposta do exercício do tópico 3

Passo 6 - script exibe_noticia.php

```
<?php

      # exibe_noticia.php

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

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

?>
```