PADRÕES DE DESENVOLVIMENTO SOLIS

14.02.2005

Diagrama

Classes

- Uma tabela é criada a partir do componente "Classe";
- Abaixo seguem as opções/especificações de cada "aba" da caixa de propriedades de uma classe.

Aba classe

- O campo "Nome da Classe" sempre é o nome da tabela;
- No campo "Comentários" pode-se informar observações sobre a tabela.

Aba atributos

- Os campos da tabela são inseridos no campo "Nome";
- No campo "Tipo" podemos informar o tipo de dado deste campo;
- No campo "Valor" podemos informar um valor padrão. Se for um valor texto, é necessário colocá-lo entre ’aspas simples’, caso seja numérico, não são necessárias as ’aspas simples’;
- Podemos utilizar o campo "Valor" para definir um campo como obrigatório(not null), partindo do princípio de que todos os campos não são obrigatórios;
- Na "Visibilidade" definimos se os campos serão:
  1. ’+’ -> ’Publico’: Campo normal;
  2. ’#’ -> ’Protegido’: Chave primária;
- Na opção "Escopo da Classe" definimos se o campo vai seguir uma seqüência, que será criada automaticamente;
- Ainda existe o campo "Comentário" que serve para fazermos comentários sobre o campo.

Aba operações

- Nesta "aba" iremos definir os índices, índices únicos, permissões e as restrições;
- Para criar um índice, digite o nome dele no campo "Nome" e no campo "Tipo" digite "index". Nos "Parâmetros", iremos definir de quais campos são os índices;
- A mesma regra serve para os índices únicos, bastando digitar a palavra "unique index" no campo "Tipo";
- Para criar uma permissão, digite "grant" no campo "Tipo". No campo "Nome", digite a permissão, all, select ou insert, por exemplo, e no campo "Parâmetro" digite para quem é essa permissão;
- As restrições funcionam da mesma forma que as permissões, bastando digitar "revoke" no campo "Tipo".

Associações

- As referências são feitas com a opção "Associação" do UML. Elas devem ligar os dois campos das tabelas e nos mostrar de quem e para quem, é a ligação. É importante fazer as ligações entre os campos exatamente nos pontos referentes a cada um dos campos da classe. Também é importante alterar as propriedades do objeto "Associação" e marcar a opção "Mostrar Seta" no "Lado B". Assim será possível identificar de quem é a referência e onde será criada a chave estrangeira.

Inserções

- As inserções são feitas com o objeto "Componente" do UML. Deve-se informar o nome da tabela e os valores para cada campo das tuplas.

Herança

- A herança é feita com o objeto "Generalização" do UML. A seta indica o pai da tabela. Ainda é possível digitar o nome do pai, caso preferir, editando a herança e preenchendo o campo "Nome" com o nome do mesmo.

Padrões de código

Tags de PHP

Utilize sempre <?php ..... ?>

Não utilize <? .... ?>

Nunca utilize <script language="PHP" ... ?> (totalmente deprecated)

Identação

Identação padrão utilizando 4 caracteres.

Configurar o editor para substituir o caracter TAB por espaços.

```
Para configurar o vim/vi:
set expandtab
set shiftwidth=4
set tabstop=4

JEdit:
Global Options->Editing
Tab width: 4
Ident width: 4
Soft tabs
```

Nome de constantes

O nome de constantes devem ser identificados com todas as letras maiúsculas.

Utilize _ para separar palavras.

Exemplo:

```
$MIOLO->....
$DB_ACAO (onde DB identifica a classe)
```

Nome de variáves, métodos, classes, atributos e funções

Iniciais de palavras, com exceção da primeira, em maiúscula.

Exemplo:

```
$personId   = "Valor";
$personName = "Valor1";

class person
{

    function listPerson()
    {

    }

}
```

Definição de funções e classes

Iniciar o bloco de código (chaves) na segunda linha.

Deixar sempre 2 quebras de linha entre os métodos e funções.

Exemplo de funções:

```
function umaFunction($arg1, $arg2 = '')
{
    if  ( condição )
    {
        código;
    }

    return $val;
}


function Connect(&$dsn, $persistent = false)
{
    if  ( is_array($dsn) )
    {
        $dsninfo = &$dsn;
    }
    else
    {
        $dsninfo = DB::ParseDSN($dsn);
    }

    if ( (! $dsninfo) || (! $dsninfo'phptype'?) )
    {
        return $this->RaiseError();
```

```
    }

    return true;
}
```

Exemplo de classes:

```
class umaClasse
{
    function umaMetodo()
    {
        codigo;
    }


    function outroMetodo()
    {
        codigo;
    }
}
```

Alinhamento

Procure sempre alinhar os sinais de "=" quando a um bloco de variáveis ou funções.

Exemplo:

```
$var1      = umaFuncao($param1, ...);
$variavel1 = outraFuncao($param1, ...);

$var2      = 10;
$variavel2 = 20;
```

Estruturas de controle

Iniciar o bloco de comandos com '{' na linha após o teste condicional.

Utilizar chaves mesmo quando existir apenas uma instrução.

No caso de teste composto com || e/ou && ... utilizar parênteses para separar os blocos e outro para delimitar todo o conjunto.

Exemplo:

```
if ( (condição1) || (condição2) )
{
      ação1;
}
elseif  ( (condição3) && (condição4) )
{
    ação2;
    ação3;
}
else
{
    ação4;
}
```

No caso do switch deixar um espaço entre os cases. Começar o bloco de código logo abaixo do case e alinhado com 2 espaços.

```
switch  ( condição )
{

    case 1:
            ação1;
            break;

    case 2:
            ação2;
            break;

    default:
            ação3;
            break;

}
```

Comentários

```
// para uma linha

/*
* comentário
* para mais de uma linha
*/
```

Composição de variável sql

As cláusulas select, from, where, and, group by, order by, etc ficam alinhadas à direita e em maiusculo, e, conseqüentemente, as identificações de arquivos, campos,... ficam alinhados à esquerda e minusculo. As condições da cláusula where também devem ter os '=' alinhados de forma que as condições fiquem alinhados à esquerda.

Exemplo:

```
    $sql = ' SELECT DISTINCT A.campo1, '.
           '                 B.campo2, '.
           '                 C.campo3, '.
           '                 D.campo1, '.
           '                 (SELECT campo '.
           '                    FROM tabela '.
           '                   WHERE id = ?), '.
           '                 count(*) '.
           '            FROM tabela1 A, '.
           '                 tabela2 B, '.
           '                 tabela3 C, '.
           '                 tabela4 D '.
           '           WHERE A.campo1 = ? '.
           '             AND B.cmp3   = ? '.
           '             AND A.campo2 = C.campo1 '.
           '             AND A.cmp4   = D.campo1 '.
           '             AND D.cmp3   = B.cmp1 '.
           '        GROUP BY C.campo1, '.
           '                 C.cmp2 ';

    $sql = ' INSERT INTO tabela1 '.
           '             (id, '.
           '              campo1, '.
           '              campo2, '.
           '              campo3) '.
           '      VALUES (?, ?, ?) ';

    $sql = ' DELETE FROM tabela1 '.
           '       WHERE campo1 = ? '.
           '         AND campo2 = ? '.
           '         AND id     = ? ';
```

```
$sql = ' UPDATE tabela1 '.
       '    SET campo1 = ? '.
       '        campo2 = ? '.
       '  WHERE id = ? ';
```

Inclusão de arquivos

Utilize o comando $MIOLO->Uses() ao invés de include_once(), require_once(), include() e require()

Este metodo e utilizado para obter acesso a outros arquivos. Quando algum arquivo nao existir, o MIOLO exibe esse problema. O metodo Uses tambem envia a informacao para o Profile e para o log.

```
MIOLO->Uses($name, $module=null);
$name (string) Nome da arquivo a ser acessado/utilizado
$module (string) Nome do modulo no qual este arquivo esta localizado. Caso esse parametro não for informado, o MIOLO tentará acessar um arquivo abaixo do diretório classes. Informando 'html_home', o MIOLO tentará acessar um arquivo abaixo do diretorio indicado pela configuração ['home']['html'] definida no arquivo miolo.conf.
```

Acesso a variáveis

Utilize MIOLO::_REQUEST ao invés de global, $GET , $POST e $REQUEST.

No MIOLO, métodos que iniciam por "_" indicam métodos estáticos, mas neste caso foi utilizado para lembrar que sua funcionalidade é similar ao $_REQUEST do PHP. O metodo _REQUEST provê uma forma mais simples para se ter acesso às variáveis. Utilizando comandos PHP, vc. teria que utilzar $_REQUEST, $_GET, $_POST ou global ao passo que no MIOLO este método tras todas as informacoes, nao importando de onde elas são provenientes. Caso você queira obter apenas o valor da variáveis provenientes de uma dessas opcoes, por exemplo GET, passe essa palavra como segundo parâmeto

```
MIOLO::_REQUEST($vars, $from='ALL');
$vars (string/array) variáveis as quais se deseja obter o valor
$from (string) de onde obter os dados. Pode ser 'GET', 'POST', além do padrão 'ALL' que retorna todos os dados.

Retorna um array com os valores das variáveis solicitadas.
```

Ambiente do MIOLO

Os programas são criados obedecendo à seguinte estrutura de diretórios;

```
|
+-modules
 |
 +-Programa
          |
          +-handlers (manipuladores: .inc)
          |
          +-db (classes Base de Dados: .class)
          |
          +-forms (classes formulários: .class)
          |
          +-classes (classes próprias do módulo/programa: .class)
          |
          +-html (arquivos com conteúdo html. geralmente arquivos de include)
          |
          +-gtk (contém os arquivos da interface php-gtk do projeto)
          |
          +-etc (quando necessário, para armazenar outros arquivos)
                |
                +-setup (arquivos relativos à configurações dos módulo)
                |
                +-sql (diretório que contém o sql de criação/alteração das bases: .sql)
                |
                +-doc (documentação)
```

Codificação dos arquivos

O arquivo principal dos sistemas deve ser main.inc, que fica localizado no diretório handlers do módulo.

Documentação

Toda a documenteção deve estar dentro de:

```
/**
*
*/
```

Escreva a documentação das classes, métodos e atributos na linha anterior e o exemplo de uso em seguida.

Tags

@author - Nome do autor que criou a classe;

@see - Gera um link para outras classes;

@since - Quando a classe ou método foi criado;

@param - Identifica um parametro de um método;

@return - Identifica o retorno de um método;

@example - Gera o exemplo de um método ou uma classe;

@deprecated - Identifica funções e métodos desatualizados que devem ser evitados;

b - coloca a em negrito;

n - quebra uma linha;

code - gera um bloco de código, o bloco termina ao encontrar um **endcode**

Exemplo

```
/**
 * Brief description of the class before dot. After the dot, we will have an more elaborated class description.
 *
 * @author Author's name [author@e.mail]
 * @author Another author [mail@e.mail]
 *
 * @version $Id$
 *
 * \b Maintainers: \n
 * Mantainer name [mantainer@e.mail.com]
 *
 * @see
 * {@link MForm}, {@link otherClass}
 *
 * @since
 * This class was created 2001/08/14
 *
 * \b Organization: \n
 * SOLIS - Cooperativa de Soluções Livres \n
 * The ??? Development Team
 *
 * \b CopyLeft: \n
 * CopyLeft (L) 2005 SOLIS - Cooperativa de Soluções Livres \n
 *
 * \b License: \n
 * Licensed under GPL (for further details read the COPYING file or http://www.gnu.org/copyleft/gpl.html )
 *
 * \b History: \n
 * See history in CVS repository: http://www.site.project.com
 *
 */
```

```
class myForm extends MForm
{
    /**
     * a public attribute.
     * Details about the $publicVar variable.
     */
    public $publicVar;

    /**
     * a private attribute.
     * Details about the $privateVar variable.
     */
    private $privateVar;


    /**
     * A constructor.
     * A more elaborate description of the constructor.
     */
    public function __construct()
    {
        code;
    }


    /**
     * A destructor.
     * A more elaborate description of the destructor.
     */
    public function __destruct()
    {
        code;
    }


    /**
     * Brief description. A member taking \b two arguments and returning an integer value.
     * @param $arg1 an integer argument.
     * @param $arg2 string argument.
     * @return (int) The results of the method
     * @see __construct()
     * @see __destruct()
     * @see sendButtonClick()
     * @see refreshData()
     */
    public function getElementValue($arg1, $arg2)
    {
        code;
    }
    /**
     * @example myForm::getElementValue An example about how to use this method
     * \code
     * <?php
     *     $form = new myForm();
     *     $form->getElementValue(1,'abc');
     *     ...
     * ?>
     * \endcode
     * You'll find another example in the file modules/tutorial3/forms/formPerson.class
     * @see {@link myForm::sendButtonClick}
     */



    /**
     * Another member. This is another important member of the class.
     * @author Author's name [author@e.mail]
     * @param $arg1 the first argument.
     * @param $arg2 the second argument.
     * @see getElementValue()
     * @since  This function was included in revision ?.?? of this file
     */
    public function sendButtonClick($arg1 , $arg2)
    {
        code;
    }


    /**
     * Old method. This is an deprecated method and should not be used.
     * @deprecated Please do not use this function, it's really deprecated.
     */
    public function anOldMethod()
    {
        code;
    }
```

```
    /**
     * Metódo não traduzido. Este método ainda não foi traduzido e dessa forma não está nos Coding Standards ;-)
     * @todo Please translate this method
     * @author Author's name [author@e.mail]
     * @since  This function was included in revision ?.?? of this file
     * @see getElementValue()
     * @return
     */
    private function anotherMethod()
    {
        code;
    }


}
```