UNIVERSIDADE DO ESTADO DO PARÁ
SPD - Serviço de Processamento de Dados

Padronização de código PHP

Belém
Setembro/2006

1. INTRODUÇÃO

O presente documento tem por objetivo apresentar diretrizes para a padronização do código escrito pelos programadores integrantes da equipe de desenvolvimento do SPD da Universidade do Estado do Pará. Esta padronização é importante no desenvolvimento de qualquer sistema, para obter ganhos em desempenho e, principalmente, para tornar mais viável a manutenção.

Visando as melhoras acima citadas, é proposta a utilização do padrão de projeto Fachada, para a melhora do desempenho e manutenção, e a padronização de nomenclatura de variáveis, para que a manutenção seja uma tarefa menos árdua.

A estrutura deste documento é bem simples, estando dividido em dois capítulos, 2. NOMENCLATURA DE VARIÁVEIS e 3. PADRÃO DE PROJETO FACHADA, como veremos a partir daqui.

2. NOMENCLATURA DE VARIÁVEIS

2.1 COMPOSIÇÃO DO NOME DE VARIÁVEIS

O nome de toda variável no código de programação deverá ser composto de: **identificador de tipo de variável** (letra minúscula) + **nome da variável** (com a letra inicial maiúscula).

2.1.1 Considerações

a) O **nome da variável** deve ser claro e, de preferência sem abreviaturas. A não ser que estas, também sejam claras.

Ex. $sMatricula em vez de $sMat (que pode se confundido com os nome matéria, matemática e etc).

b) Se o **nome da variável** for composto por dois nomes e **não** for uma variável que figura dentro de alguma classe, ou **não** servirá de parâmetro para a chamada de métodos, então o **nome da variável** será composto pelo **primeiro nome com a inicial maiúscula** + **segundo nome com a inicial maiúscula**

c) Se o **nome da variável** for composto por dois nomes e for uma variável que figura dentro de alguma classe, ou servirá de parâmetro para a chamada de métodos, então o **nome da variável** será composto pelo **primeiro nome com a inicial maiúscula** + **segundo nome com a inicial minúscula**

2.2 IDENTIFICADOR DE TIPO DE VARIÁVEL

O identificador de tipo de variável, como já está explicito no nome, indica de que tipo é a variável em questão, e identifica os tipos String, Numérico, Objeto, Boleano e Vetor, como podemos notar na tabela a seguir.

TIPO	IDENTIFICADOR
Boleano	“b”
String	“s”
Numérico	“n”
Objeto	“o”

Como se pode observar, o tipo Vetor não figura na tabela acima, isso porque a nomenclatura depende do tipos e dados armazenados dentro do vetor como demonstra a tabela abaixo:

TIPO DE VETOR	IDENTIFICADOR
Vetor de dados boleanos	“vb”
Vetor de dados string	“vs”
Vetor de dados numéricos	“vn”
Vetor de objetos	“vo”
Vetor de vetores (Matriz)	“vv”
Vetor com dados de tipos diversos	“v”

2.3 EXEMPLOS:

a) Vetor de objetos da classe Matricula → $voMatricula;

b) Variável que representa o retorno verdadeiro ou falso de um função → $bResultado;

c) Variável que contem um valor de uma nota → $nNota;

d) Variável que contem um nome de aluno → $sNomeAluno;

e) Variável de classe ou usada como parâmetro no método da classe que contem um nome de aluno → $sNomealuno;

3. PADRÃO DE PROJETO FACHADA

O objetivo aqui, não é explicar a teoria deste padrão de projeto, e sim explicar como este está sendo (será) utilizado, na prática, dentro da instituição.

3.1 FUNCIONAMENTO

O padrão de projeto Fachada, da maneira como está sendo (será) utilizado na UEPA, funciona da seguinte forma:

São utilizadas classes com funções distintas, sendo elas: a classe Simples, responsável por operações que não interagem com o banco de dados; a classe BD, que realiza as interações com o banco de dados; a classe BancoDeDados, que realiza todas as operações necessárias do sistema no banco de dados, servindo de interface entre a classe BD e o servidor de banco; e a classe Fachada, que agrupa os métodos de todas as classes Simples e BD que possuem um objetivo final comum.

Uma vantagem de se usar a Fachada é que com um único objeto desta classe, poderão ser chamados vários métodos que manipulam diversas informações para um determinado fim.

Outra vantagem deste padrão não está propriamente no código ou na forma como se escreve este código, mas sim no programa desenvolvido pela equipe do SPD, o Gera Classes, que gera todas as classes necessárias para o funcionamento do sistema a partir das tabelas do banco de dados, poupando assim um trabalho muito grande.

Nos tópicos a seguir vamos mostrar como funciona o Gera Classes, o padrão em que ele gera as classes, e os métodos padrão de cada tipo de classe gerada.

3.2 O GERA CLASSES

O Gera Classes, como já dissemos, tem o objetivo de escrever todas as classes necessárias ao funcionamento do sistema, com métodos pré-definidos para cada classe, que poderão ser alterados posteriormente. Ainda poderão ser implementados novos métodos. Antes de explicarmos o padrão das classes geradas, explicaremos a classe BancoDeDados, que já é padrão e já está definida no sistema.

3.2.1 A classe BancoDeDados

Para melhor ser entendido o funcionamento da classe, explicaremos método por método.

a) Método ***iniciaConexaoBanco()*** → responsável por fazer a conexão com o Banco de Dados. O usuário e a senha serão os que estiverem registrados na sessão.

b) Método ***terminaConexaoBanco()*** → responsável por fechar a conexão com o Banco de Dados.

c) Método ***insereRegistroNoBanco($sTabela,$sCampos,$sValores)*** → responsável por inserir registros no banco de dados, recebendo os parâmetros para gerar a sentença SQL, sendo: $sTabela a tabela onde será inserido o registro; $sCampos os campos da tabela que receberão valores; e $sValores os valores recebidos pelos campos. É importante que as strings sejam passadas na ordem correta para que a sentença seja eficaz. A sentença gerada será "INSERT INTO $sTabela ($sCampos) VALUES ($sValores) ".

d) Método ***recuperaRegistrosDoBanco($sCampos,$sTabelas,$sComplemento)*** → responsável por recuperar registros no banco de dados, recebendo os parâmetros para gerar a sentença SQL, sendo: $sTabelas as tabelas das quais serão extraídos os registros para a consulta; $sCampos os campos da tabela que serão retornados; e $sComplemento as cláusulas para a consulta no banco de dados. É importante que as strings sejam passadas na ordem correta para que a sentença seja eficaz. A sentença gerada será "SELECT $sCampos FROM $sTabela $sComplemento ".

e) Método ***alteraRegistrosDoBanco($sTabela,$sCampos,$sComplemento)*** → responsável por alterar um registro do banco de dados, recebendo os parâmetros para gerar a sentença SQL, sendo: $sTabela a tabela em que o registro será alterado; $sCampos os campos da tabela que serão alterados; e $sComplemento as cláusulas para a alteração no banco de dados. É importante que as strings sejam passadas na ordem correta para que a sentença seja eficaz. A sentença gerada será "UPDATE $sTabela SET $sCampos $sComplemento".

f) Método ***excluiRegistrosDoBanco($sTabela,$sComplemento)*** → responsável por excluir um registro do banco de dados. Na verdade, o método apenas seta o atributo ativo da Tabela para *false*, ficando totalmente obsoleto caso a tabela não possua campo ativo. Este método os parâmetros para gerar a sentença SQL, sendo: $sTabela a tabela em que o registro será alterado; $sComplemento as cláusulas para a alteração no banco de dados. É importante que as strings sejam passadas na ordem correta para que a sentença seja eficaz. A sentença gerada será "UPDATE $sTabela SET ATIVO = FALSE $sComplemento".

g) Método ***excluiFisicamenteRegistrosDoBanco($sTabela,$sComplemento)*** → responsável por excluir realmente um registro do banco de dados, utilizando os parâmetros para gerar a sentença SQL, sendo: $sTabela a tabela em que o registro será deletado; $sComplemento as cláusulas para a exclusão no banco de dados. É importante que as strings sejam passadas na ordem correta para que a sentença seja eficaz. A sentença gerada será "DELETE FROM $sTabela $sComplemento".

h) Método ***retornaCampo($sTabela)*** → responsável por recuperar todos os campos de uma tabela, utilizando os parâmetros para gerar a sentença SQL, sendo: $sTabela a tabela que se deseja obter os campos. O retorno deste método é uma matriz que em cada linha possui a posição 'sNome', que guarda o nome do campo, e a posição 'sTipo', que guarda o tipo do campo.

i) Método ***retornaTabela()*** → responsável por recuperar todas as tabelas do banco de dados. O retorno deste método é um vetor que guarda os nomes das tabelas.

j) Método ***retornaChavePrimaria($sTabela)*** → responsável por recuperar todos os campos que são chaves-primária de uma tabela, utilizando os parâmetros para gerar a sentença SQL, sendo: $sTabela a tabela que se deseja obter os campos primários. O retorno deste método é uma matriz que em cada linha possui a posição 'sNome', que guarda o nome do campo primário, e a posição 'sTipo', que guarda o tipo do campo primário.

Explicada a classe BancoDeDados, vamos partir para as classes geradas pelo Gera Classes, que são as classes simples, as classes BD e a Fachada.

3.2.2 A classe Simples

Como já foi dito, uma classe simples será gerada para cada tabela do banco de dados. O nome da classe o será o próprio nome da tabela, com a primeira letra maiúscula e as demais minúsculas. Para cada campo da tabela será gerado um atributo dentro da classe. Os métodos padrão são os métodos construtor e get() e set(). O método construtor atribui valores para cada atributo da classe. E os métodos get() e set() servem para retornar e atribuir valor para cada atributo, respectivamente. Abaixo está um exemplo de uma classe simples gerada para a tabela bairro (idbai,descbai,idmun,codbaisisca):

```
<?
 class Bairro{
        var $sDescbai;
        var $nIdbai;
        var $nIdmun;
        var $nCodbaisisca;

        function Bairro($sDescbai,$nIdbai,$nIdmun,$nCodbaisisca){
                $this->setDescbai($sDescbai);
                $this->setIdbai($nIdbai);
                $this->setIdmun($nIdmun);
                $this->setCodbaisisca($nCodbaisisca);

        }

        function setDescbai($sDescbai){
                $this->sDescbai = $sDescbai;
        }
        function getDescbai(){
                return $this->sDescbai;
        }
        function setIdbai($nIdbai){
                $this->nIdbai = $nIdbai;
        }
        function getIdbai(){
                return $this->nIdbai;
        }
        function setIdmun($nIdmun){
                $this->nIdmun = $nIdmun;
        }
        function getIdmun(){
                return $this->nIdmun;
        }
        function setCodbaisisca($nCodbaisisca){
                $this->nCodbaisisca = $nCodbaisisca;
        }
        function getCodbaisisca(){
                return $this->nCodbaisisca;
        }

 }
 ?>
```

3.2.3 A classe BD

Também para cada tabela do banco de dados será gerada uma classe BD, cujo nome será o será o próprio nome da tabela, com a primeira letra maiúscula e as demais minúsculas, acrescido das letras BD no final. A classe BD tem por objetivo fazer as interações com o banco de dados através da classe BancoDeDados e possui por padrão os seguintes métodos:

a) Método ***inserir($oObjeto)*** → responsável por inserir um objeto da classe Simples no banco de dados, recebendo como parâmetro o objeto de uma classe. Com os atributos deste objeto serão construídas as variáveis necessárias para a inserção no banco de dados através da classe BancoDeDados. O método retorna verdadeiro caso a inserção seja feita com sucesso, ou falso caso haja fracasso.

b) Método ***alterar($oObjeto)*** → responsável por alterar um registro no banco de dados com os valores dos atributos da classe simples, recebendo como parâmetro o objeto de uma classe. Com os atributos deste objeto serão construídas as variáveis necessárias para a alteração no banco de dados através da classe BancoDeDados. O método retorna verdadeiro caso a alteração seja feita com sucesso, ou falso caso haja fracasso.

c) Método ***presente($atributos-chave)*** → responsável por verificar a existência de um registro da tabela no banco de dados, recebendo como parâmetro os atributos-chave do registro que se deseja fazer a verificação. Com os atributos-chave serão construídas as variáveis necessárias para a consulta no banco de dados através da classe BancoDeDados. O método retorna verdadeiro caso se comprove a existência do registro, ou falso em caso contrário.

d) Método ***recuperaTodos()*** → responsável por recuperar todos os registros da tabela no banco de dados. Neste método ainda serão construídas as variáveis necessárias para a consulta no banco de dados através da classe BancoDeDados. O método retorna um vetor que guarda todos os objetos que representam registros do banco, caso haja registro na tabela, ou falso, caso a tabela esteja vazia.

e) Método ***recuperaUm($atributos-chave)*** → responsável por recuperar um registro da tabela no banco de dados, recebendo como parâmetro os atributos-

chave do registro. Com os atributos-chave serão construídas as variáveis necessárias para a consulta no banco de dados através da classe BancoDeDados. O método retorna um objeto de classe simples que representa o registro, caso haja o registro com os parâmetros enviados, ou falso, em caso contrário.

f) Método ***excluir($atributos-chave)*** → responsável por excluir um registro da tabela no banco de dados, recebendo como parâmetro os atributos-chave do registro que se deseja fazer a exclusão. Com os atributos-chave serão construídas as variáveis necessárias para a sentença SQL no banco de dados através da classe BancoDeDados. O método retorna verdadeiro caso a exclusão seja feita com sucesso, ou falso em caso contrário.

3.2.4 A classe Fachada

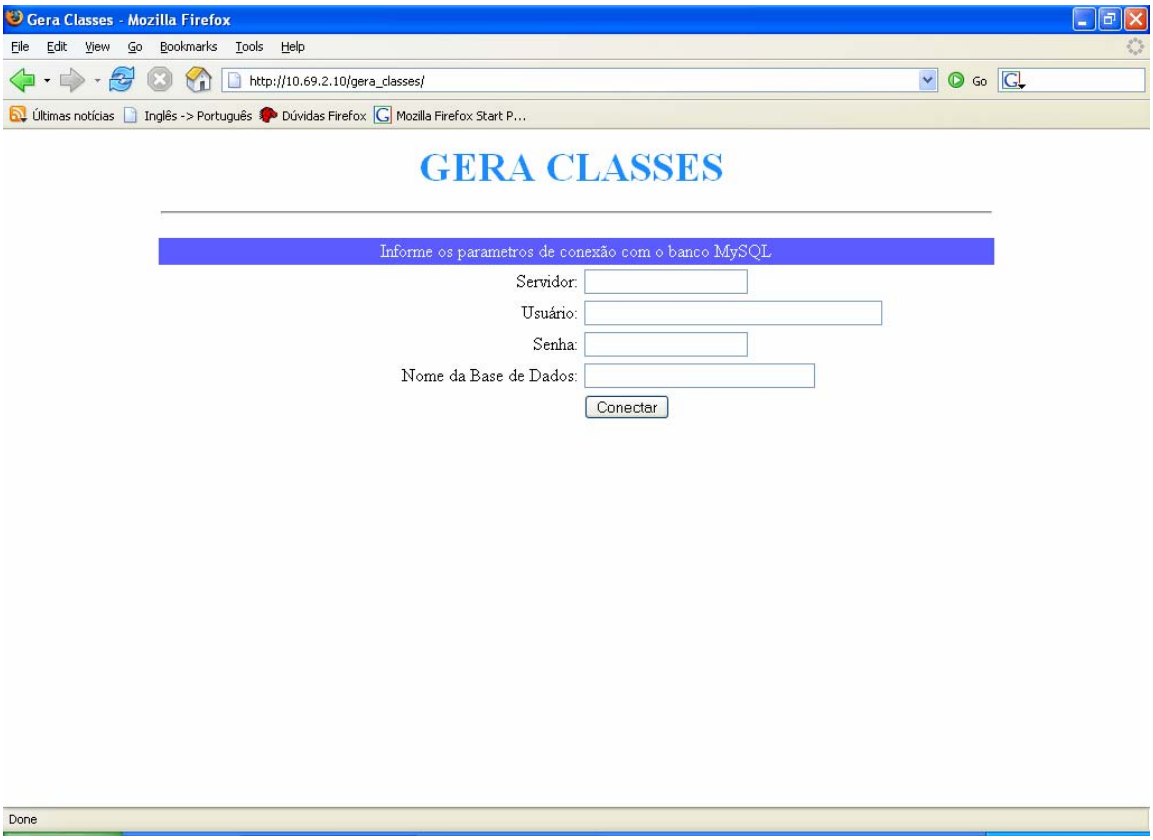
A classe fachada é o agrupamento de classes com objetivo final comum, e seu nome é escolhido no ato da geração das classes, sendo **Fachada + nome escolhido + BD.**

Nesta classe, serão chamados todos os métodos de cada classe BD descritos acima, além de métodos construtores de objetos de classe simples e de classe BD. Para não haver redeclaração de métodos, já que os métodos são padrão, na fachada cada método será constituído por seu **nome + classe a qual pertence.** Por exemplo, o método *excluir* da classe Bairro, se chamará na Fachada *excluirBairro,* e o método *recuperaTodos* da classe Nota será chamado de *recuperaTodosNota* na Fachada, e assim sucessivamente.

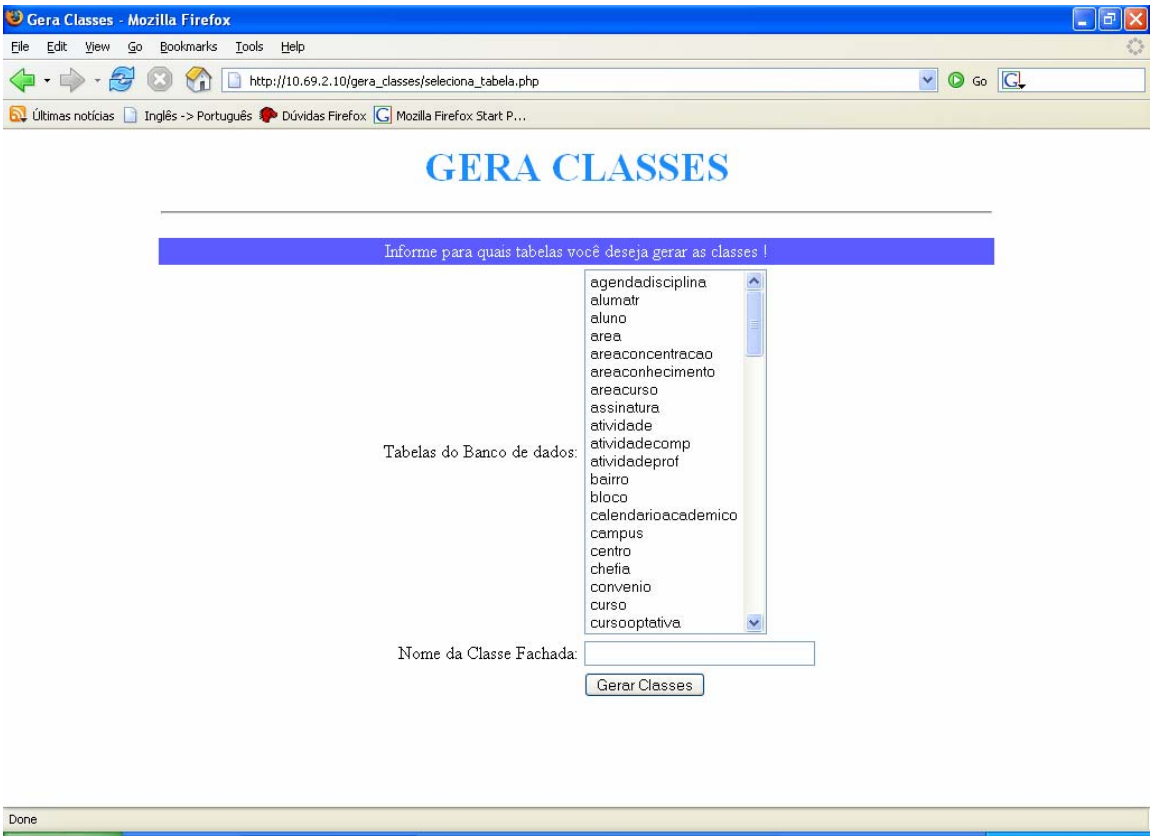
Esta classe apenas faz a chamada para os métodos das classes, por isso recebe os mesmos parâmetros e retorna os mesmos valores em cada método explicado no tópico acima.

3.3 GERANDO CLASSES

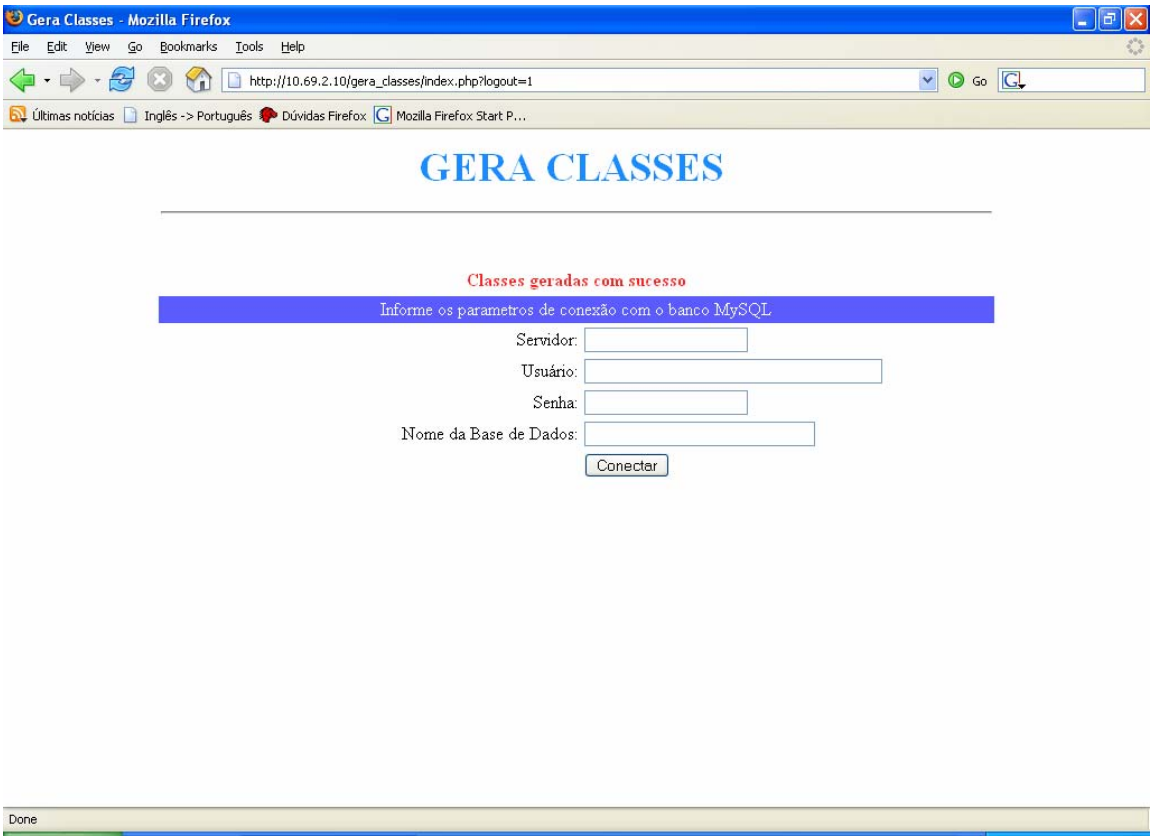
Para acessar o programa gerador de classes, deve-se digitar na barra de endereços do browser o seguinte endereço: **http://10.69.2.10/gera_classes/.** Chegando assim na página inicial vista abaixo:



É na tela inicial que será feita a conexão com o banco de dados. O usuário deve fornecer o endereço do servidor de banco de dados, o usuário de banco, a senha deste usuário, e o nome da base de dados. Se a conexão for feita com sucesso, será mostrada a tela abaixo.



Nesta tela serão escolhidas as tabelas para as quais serão geradas as classes simples, as classes BD e a Fachada, e o nome da Fachada. As várias tabelas podem ser selecionadas segurando a tecla [Ctrl], e clicando sobre as mesmas. O nome da Fachada deve iniciar com letra maiúscula, com as demais minúsculas, para cada nome que compõem o nome da Fachada. Se as classes forem geradas com sucesso a partir das tabelas selecionadas, o programa retorna para tela inicial com uma mensagem de sucesso como mostra a figura a seguir:



Se o usuário chegou nesta tela, então as classes estão geradas e gravadas numa pasta do servidor chamada **classes_geradas**, que fica dentro da pasta do programa **gera_classes**. Através de ftp, é possível baixar as classes que foram geradas.

Obs.: são geradas junto com as classes, páginas de processamento para cada classe, mas como o padrão da UEPA é diferente, estas páginas ficam obsoletas. Mas se você quise baixar para olhar e ter uma referência, elas ficam dentro da pasta **paginas_geradas** no mesmo local da pasta **classes_geradas**.

4. CONCLUSÃO

Este documento é apenas um guia na padronização de código PHP para a equipe de desenvolvimento do SPD da UEPA, podendo, e devendo, ser alterado com melhorias e novos padrões que possam surgir.

Mas o importante é que se tente seguir o padrão o máximo possível, para que haja entendimento entre os programadores atuais e os novos programadores que poderão chegar. Tornando assim, mais fácil a tarefa de manutenção do sistema.