DEFINICIÓN ESTÁNDAR DE CODIFICACIÓN

Universidad Tecnológica de la Riviera Maya.

Por

Ramírez López Miguel Ángel.

Materia: Calidad en el Desarrollo del Software

Grupo: TI-51.

Contenido

DEFINICIÓN ESTÁNDAR DE CODIFICACIÓN

La estructura del proyecto está realizada en el esquema MVC, utilizando el estándar de Zend Framework y cumpliendo los requerimientos de phpDocumentor.

Nombre de los archivos

Para cualquier otro archivo, sólo caracteres alfanuméricos, barras bajas (_) y guiones (-) están permitidos. Los espacios en blanco están estrictamente prohibidos.

Cualquier archivo que contenga código PHP debe terminar con la extensión " .php ", con la excepción de los scripts de la vista. Los siguientes ejemplos muestran nombres de archivo admisibles para clases de Zend Framework.

```
Zend/Db.php

Zend/Controller/Front.php

Zend/View/Helper/FormRadio.php
```

Los nombres de archivo deben apuntar a nombres de clases como se describe arriba.

Variables

Los nombres de variables pueden contener caracteres alfanuméricos. Las barras bajas (_) no están permitidas. Los números están permitidos en los nombres de variable pero no se aconseja en la mayoría de los casos.

Para las variables de instancia que son declaradas con el modificador "private" o "protected", el primer carácter de la variable debe ser una única barra baja (_). Este es el único caso admisible de una barra baja en el nombre de una variable. Las variables declaradas como "public" no pueden empezar nunca por barra baja.

Al igual que los nombres de funciones, los nombres de variables deben empezar siempre con una letra en minúscula y seguir la convención "camelCaps".

Por norma general, se recomienda la elocuencia. Las variables deberían ser siempre tan elocuentes como prácticas para describir los datos que el desarrollador pretende almacenar en ellas. Variables escuetas como " $i " y " $n " están desaconsejadas, salvo para el contexto de los bucles más pequeños. Si un bucle contiene más de 20 líneas de código, las variables de índice deberían tener nombres más descriptivos.

Constantes

Las constantes pueden contener tanto caracteres alfanuméricos como barras bajas (_). Los números están permitidos. Todas las letras pertenecientes al nombre de una constante deben aparecer en mayúsculas.

Las palabras dentro del nombre de una constante deben separarse por barras bajas (_). Por ejemplo, EMBED_SUPPRESS_EMBED_EXCEPTION está permitido, pero EMBED_SUPPRESSEMBEDEXCEPTION no.

Las constantes deben ser definidas como miembros de clase con el modificador "const". Definir constantes en el alcance global con la función "define" está permitido pero no recomendado.

Estructuras de control

Las sentencias de control basadas en las construcciones *if* y *elseif* deben tener un solo espacio en blanco antes del paréntesis de apertura del condicional y un solo espacio en blanco después del paréntesis de cierre.

Dentro de las sentencias condicionales entre paréntesis, los operadores deben separarse con espacios, por legibilidad. Se aconseja el uso de paréntesis internos para mejorar la agrupación lógica en expresiones condicionales más largas.

La llave de apertura "{" se escribe en la misma línea que la sentencia condicional. La llave de cierre "}" se escribe siempre en su propia línea. Cualquier contenido dentro de las llaves debe separarse con cuatro espacios en blanco.

```
if ($a != 2) {
    $a = 2;
}
```

Si la sentencia condicional hace que la longitud de la línea supere el máximo establecido y tiene varias cláusulas, es posible romper el condicional en varias líneas. En tal caso, romper la línea antes de un operador lógico, y el paréntesis de la línea de tal manera que se coloque bajo el primer carácter de la cláusula condicional. Los paréntesis de cierre en el condicional a continuación, se colocan en una línea con la llave de apertura, con un solo espacio que separa los dos, a un nivel de sangría equivalente a la sentencia de control de apertura.

```
if (($a == $b)
    &amp;&amp; ($b == $c)
    || (Foo::CONST == $d)
) {
    $a = $d;
}
```

Para las declaraciones "if" que incluyan "elseif" o "else", las convenciones de formato son similares a la construcción "if". Los ejemplos siguientes demuestran el formato correcto para declaraciones "if" con construcciones "else" y/o "elseif":

```
if ($a != 2) {
    $a = 2;
} else {
    $a = 7;
}

if ($a != 2) {
    $a = 2;
} elseif ($a == 3) {
    $a = 4;
} else {
    $a = 7;
}

if (($a == $b)
    &amp;&amp; ($b == $c)
    || (Foo::CONST == $d)
```

```
) {
    $a = $d;
} elseif (($a != $b)
          || ($b != $c)
) {
    $a = $c;
} else {
    $a = $b;
}
```

PHP permite escribir sentencias sin llaves -{}- en algunas circunstancias. Este estándar de código no hace ninguna diferenciación- toda sentencia "if", "elseif" o "else" debe usar llaves.

El uso de la construcción "elseif" está permitido pero no se aconseja, en favor de la combinación "else if".

Switch

Las declaraciones de control escritas con la declaración "switch" deben tener un único espacio en blanco antes del paréntesis de apertura del condicional y después del paréntesis de cierre.

Todo contenido dentro de una declaración "switch" debe separarse usando cuatro espacios. El contenido dentro de cada declaración "case" debe separarse usando cuatro espacios adicionales.

```
switch ($numPeople) {
    case 1:
        break;

    case 2:
        break;

    default:
        break;
}
```

La construcción default no debe omitirse nunca en una declaración switch.

Clases

Zend Framework se estandariza una convención de nombres de clases donde los nombres de las clases apuntan directamente a las carpetas en las que estan contenidas. La carpeta raíz de la biblioteca estándar de Zend Framework es la carpeta "Zend/", mientras que la carpeta raíz de las bibliotecas extra de Zend Framework es la carpeta "ZendX/". Todas las clases de Zend Framework están almacenadas jerárquicamente bajo estas carpetas raíz.

Los nombres de clases pueden contener sólo caracteres alfanuméricos. Los números están permitidos en los nombres de clase, pero desaconsejados en la mayoría de casos. Las barras bajas (_) están permitidas solo como separador de ruta (el archivo " Zend/Db/Table.php " debe apuntar al nombre de clase " Zend_Db_Table ").

Si el nombre de una clase está compuesto por más de una palabra, la primera letra de cada palabra debe aparecer en mayúsculas. Poner en mayúsculas las letras siguientes no está permitido, ej: "Zend_PDF" no está permitido, mientras que " Zend_Pdf " es admisible.

Estas convenciones definen un mecanismo de pseudo-espacio de nombres para Zend Framework. Zend Framework adoptará la funcionalidad PHP de espacio de nombres cuando esté disponible y sea factible su uso en las aplicaciones de nuestros desarrolladores.

Vea los nombres de clase en las bibliotecas estándar y adicionales (extras) como ejemplos de esta convención de nombres.

Clases Abstractas

En general, las clases abstractas siguen las mismas convenciones que las clases, con una regla adicional: Los nombres de las clases abstractas deben acabar con el término, "Abstract", y ese término no debe ser precedida por un guión bajo. Ejemplo, `Zend_Controller_Plugin_Abstract` es considerado un nombre no válido, pero `Zend_Controller_PluginAbstract` o `Zend_Controller_Plugin_PluginAbstract` serían nombres válidos.

Métodos

Los nombres de funciones pueden contener únicamente caracteres alfanuméricos. Las guiones bajos (_) no estan permitidos. Los números están permitidos en los nombres de función pero no se aconseja en la mayoría de los casos.

Los nombres de funciones deben empezar siempre con una letra minúscula. Cuando un nombre de función consiste en más de una palabra, la primera letra de cada nueva palabra debe estar en mayúsculas. Esto es llamado comúnmente como formato "camelCase".

Por norma general, se recomienda la elocuencia. Los nombres de función deben ser lo suficientemente elocuentes como para describir su propósito y comportamiento.

Estos son ejemplos de nombres de funciones admisibles:

```
filterInput()
```

```
getElementById()
```

```
widgetFactory()
```

Para la programación orientada a objetos, los métodos de acceso para las instancias o variables estáticas deben ir antepuestos con un "get" o un "set". Al implementar el patron de diseño, tales como el patrón singleton o el patrón factory, el nombre del método deben contener en la práctica el nombre del patrón para describir su comportamiento de forma más completa.

Para el caso en que los métodos son declarados con el modificador "private" o "protected", el primer carácter del nombre de la variable debe ser una barra baja (_). Este es el único uso admisible de una barra baja en un nombre de método. Los métodos declarados como públicos no deberían contener nunca una barra baja.

Las funciones de alcance global (también llamadas "funciones flotantes") están permitidas pero desaconsejadas en la mayoría de los casos. Considere envolver esas funciones en una clase estática.

Formato de documentación

Todos los archivos de clase deben contener un bloque de documentación "a nivel de archivo" al principio de cada archivo y un bloque de documentación "a nivel de clase" inmediatamente antes de cada clase. Ejemplo de estos bloques de documentación pueden encontrarse debajo.

Archivos

Cada archivo que contenga código PHP debe tener un bloque de documentación al principio del archivo que contenga como mínimo las siguientes etiquetas phpDocumentor:

```
/**
 * Descripción corta del fichero
 *
 * Descripción larga del fichero (si la hubiera)...
 *
 * LICENSE: Some license information
 *
 * @category   Zend
 * @package    Zend_Magic
 * @subpackage Wand
 * @copyright  Copyright (c) 2005-
2011 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license   BSD License
 * @version    $Id:$
 * @link       http://framework.zend.com/package/PackageName
 * @since      File available since Release 1.5.0
*/
```

Clases

Cada clase debe contener un bloque de documentación que contenga como mínimo las siguientes etiquetas phpDocumentor:

```
/**
 * Descripción corta de la clase
 *
 * Descripcion larga de la clase (si la hubiera)...
 *
 * @category   Zend
 * @package    Zend_Magic
 * @subpackage Wand
 * @copyright  Copyright (c) 2005-
2011 Zend Technologies USA Inc. (http://www.zend.com)
 * @license    http://framework.zend.com/license   BSD License
 * @version    Release: @package_version@
 * @link       http://framework.zend.com/package/PackageName
 * @since      Class available since Release 1.5.0
 * @deprecated Class deprecated in Release 2.0.0
 */
```

Funciones

Cada función, incluyendo métodos de objeto, debe contener un bloque de documentación que contenga como mínimo:

- Una descripción de la función
- Todos los argumentos
- Todos los posibles valores de retorno

No es necesario incluir la etiqueta "@access" si el nivel de acceso es conocido de antemano por el modificador "public", "private", o "protected" usado para declarar la función.

Si una función/método puede lanzar una excepción, utilice @throws para todos los tipos de excepciones conocidas:

```
@throws exceptionclass [description]
```

Bibliografía

Framework, Z. (s.f.). Obtenido de http://manual.zfdes.com/es/coding-standard.html