

CONVENCIONES DE CODIFICACION

En cualquier proyecto de desarrollo de sistemas se hace necesario seguir unas guías comunes para asegurar una correcta comprensión del código fuente así como su posterior mantenimiento por personas que no han participado del desarrollo inicial. La disparidad de estilos trae consigo un efecto negativo en la salud del proyecto; al inicio aparentan ser productivas, ingeniosas, eficaces, pero a largo plazo cuando llega el momento de corregir errores que ocurren en entornos o condiciones no previstas, o cuando se deben realizar ampliaciones del sistema, o cuando se necesita incorporar un nuevo programador; es donde se pueden ver los efectos negativos de las malas prácticas.

La convención de nombres es un conjunto de normas y reglas para la escritura de nombres, código fuente, identificadores, comentarios u otros elementos dentro de la programación de sistemas, que facilitan y hacen más comprensible no solamente su lectura sino también su comprensión.

Lineamientos generales

- Salvo necesidades especiales se convendrá que la codificación de los nombres sean escritas en *español*, exceptuando aquellas que tienen origen externo pero que son incluidas en nuestros programas por los beneficios que estos describan (librerías, componentes, frameworks, etc.)
- En los nombres de identificadores queda prohibido el uso de:
 - Determinantes:
 - ✓ Artículos: el, la, los, las, uno, unos, unas,
 - ✓ Determinantes demostrativos: este, ese, aquel,
 - ✓ Determinantes posesivos: mi, tu, su, cuyo, cuyos, cuyas,
 - ✓ Cardinales: uno, dos, tres,
 - ✓ Ordinales: segundo, tercero, cuarto,
 - ✓ Multiplicativos: triple, cuádruple, quíntuple,

- Pronombres:

- ✓ Personales: yo, tu, el ,
- ✓ Recíprocos: os, nos, se,
- ✓ Reflexivos: me, te, se,
- ✓ Interrogativos/Exclamativos: que, como, cual, cuanto,

Las únicas excepciones a estas reglas son el ordinal “*Primer*” y el multiplicativo “*Doble*”

- Evitar el uso de preposiciones, exceptuando situaciones de marcada necesidad: a, ante, bajo, con, de, desde, en, hacia, hasta, para,
- En todo momento se utilizaran nombres que sean claros, concretos, expresivos y no ambiguos. Por ejemplo: fechaTransaccion vs fecha, estadoCorrespondencia vs estado, etc.

Codificación Camel Case

CamelCase es un estándar en varios lenguajes de programación. Es la práctica de escribir frases o palabras compuestas eliminando los espacios y poniendo en mayúscula la primera letra de cada palabra. El nombre se deduce al parecido de estas mayúsculas, a las jorobas de los camellos.

Existen dos tipos de CamelCase:

UpperCamelCase: la primera letra de todas se escribe en mayúscula al igual que las primeras letras de las restantes palabras. Por ejemplo:

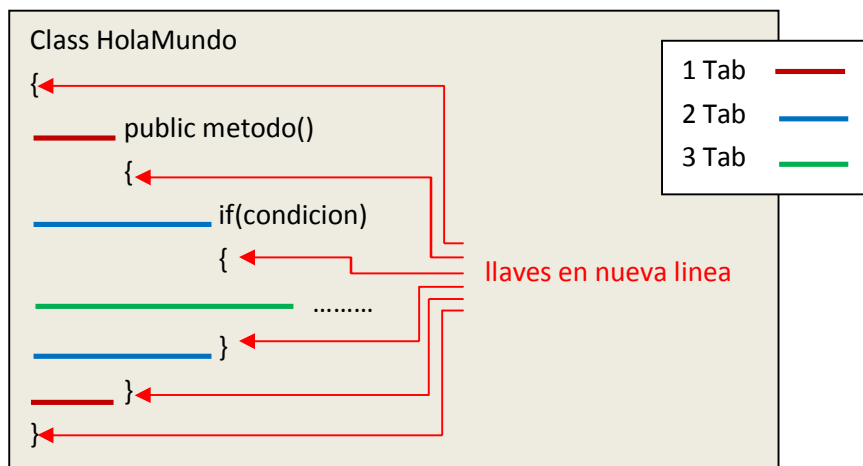
NombreClase

lowerCamelCase: la primera letra de la primer palabra se escribe en minúscula y las primeras letras de las restantes palabras están escritas en mayúsculas. Por ejemplo:

nombreObjeto

Indentación

Para la indentación existen varios estilos tales como: *Allman*, *K&R*, *BSD KN*, *Whitesmiths*, etc. Por convención se utilizará el **estilo Allman**, el cual dice que debemos usar los sangrados (tabulación) para **indentar** el código, nunca espacios y colocar las llaves de control en la línea subsiguiente.



¿Indentar seteo de variables?

Esto no es un estándar, pero puede tomarse como una buena recomendación. Por ejemplo en PHP se tiene:

antes

```
$nombreTemporal = $_FILES['archivo']['nombreTemporal'];
$tamanoArchivo = $_FILES['archivo']['tamano'];
$nombreReal = $_FILES['archivo']['nombre'];
```

después (con indentación)

```
$nombreTemporal    = $_FILES['archivo']['nombreTemporal'];
$tamanoArchivo      = $_FILES['archivo']['tamano'];
$nombreReal         = $_FILES['archivo']['nombre'];
```

Sin duda se lee mejor de la segunda manera, queda más claro y se distinguen mejor las variables. También podemos hacer lo mismo con los parámetros de funciones cuando las líneas se repiten:

```
$sitio->configuracion("cabecera"      , $cabecera);  
$sitio->configuracion("menu"          , $menu);  
$sitio->configuracion("descripcion"    , $descripcion);  
$sitio->configuracion("titulo"         , $titulo);
```

Tamaño máximo de línea

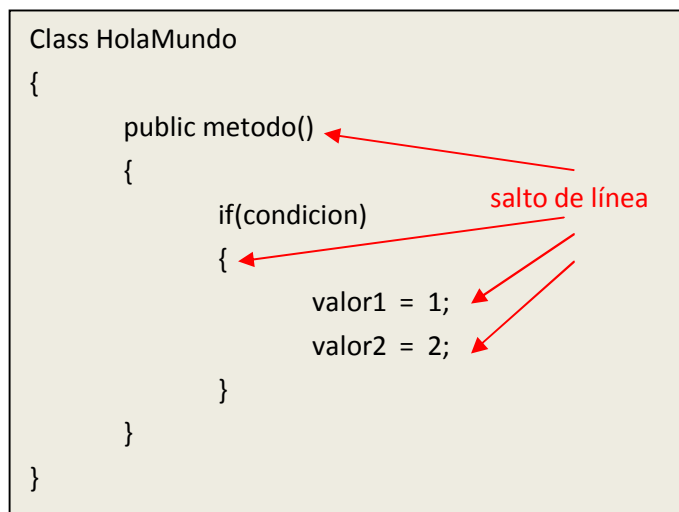
Evitar que la longitud de la línea no supere los 80 caracteres (esto no es restrictivo). Cuando se supera esta longitud se debe cortar bajo los siguientes principios:

- Salto de línea después de la coma
- Salto de línea después de un operador
- Alinear la nueva línea con el principio de la expresión en el mismo nivel que la línea anterior

Salto de línea

Añadir un salto de línea después del cierre de los paréntesis de los parámetros, así también después de un punto y coma o cuando termine una sentencia.

```
Class HolaMundo  
{  
    public metodo()  
    {  
        if(condicion)  
        {  
            valor1 = 1;  
            valor2 = 2;  
        }  
    }  
}
```



Espacios y líneas en blanco

- Usar espacios en blanco para mejorar la legibilidad del código
- Usar espacios en blanco en ambos lados del operador de símbolos, después de comas y después de las declaraciones

- Usar líneas en blanco para separar trozos de código
- Usar líneas en blanco antes de cada método dentro de la clase

```
Class HolaMundo
{
    public metodo(parametro1, parametro2)
    {
        //comentarioA
        if(condicionA)
        {
            valor1 = 1;
            valor2 = 2;
        }

        //comentarioB
        if(condicionB)
        {
            valorB = 1 + valor1;
        }
    }
}
```

Después de una coma

separación de código

separar operadores

Nombres de clases

Para el nombre de clases tomar en cuenta:

- Las clases representan “cosas” y no “acciones”, por tal motivo evitar los verbos.
- El nombre debe definirse en “singular”, salvo que la clase represente multiplicidad de cosas.
- Los nombres de las clases deben ser “sustantivos”. Por ejemplo Automovil, Persona, Pais, Armamento, Empresa
- Cada clase debe tener un bloque de documentación según la norma del lenguaje (phpDocumentor para PHP, javaDoc para Java, etc.).

En la definición del nombre de clases se aplicara el estilo CamelCase. Solo deberían contener caracteres alfanuméricos incluyendo el carácter underscore (_). Tal nombre debe ser significativo y

brindar una idea de lo que representa. La primera letra siempre debe ir en mayúscula (UpperCamelCase). Por ejemplo:

definición incorrecta: clase_ejemplo, nombreClase

definición correcta: NombreClase, Nombre_Clase

Interfaces

Las interfaces seguirán las mismas reglas de las clases con la única exigencia que deben añadirse en la terminación la palabra *Interface*. Por ejemplo:

BaseDatosInterface

Funciones, Procedimientos y Métodos

Las funciones y métodos deben estar en *lowerCamelCase*, utilizando en caso necesario el caracter underscore (_) para separar palabras brindando el sentido completo y humano de su funcionamiento. Un buen nombre para una rutina es aquel que describe todo lo que hace la rutina; bajo ningún caso utilizar verbos genéricos. Por ejemplo: procesarActivo, gestionarFuncionario, etc. El nombre debe consistir en un verbo seguido del objeto al que afecta. Por ejemplo:

imprimirFichaActivo

modificarDatosFuncionario

calcularPromedioPonderado

Parámetros

Los parámetros de rutinas deben estar ordenados de la siguiente manera:

- Primero los parámetros de entrada o solo lectura.
- Después los parámetros de transporte o de lectura/escritura
- Finalmente, los parámetros exclusivamente de salida

No se deben utilizar los parámetros como variables comunes (auxiliares, contadores, temporales, generales, etc.). Por ejemplo el siguiente fragmento de código es inapropiado:

```
function funcionCalculoEjemplo($entrada, $salida)
{
    $entrada = $entrada + funcionOtroCalculo($entrada);
    $entrada = $entrada + ($entrada / 2);
    .....
    $salida  = $entrada;
}
```

La única situación en la que se permite la modificación de los parámetros de entrada es para normalizar su valor como preparación a su uso.

```
function funcionEjemplo($codigoBarra)
{
    $codigoBarra = trim( strtoupper($codigoBarra));
    .....
}
```

Variables

Cualquier tipo de variables deben estar escritas en *lowerCamelCase* y cuando una variable representa una instancia de una clase debe tener el mismo nombre de la clase con el prefijo *obj* haciendo alusión a la creación del objeto correspondiente, también en *lowerCamelCase*; y de forma similar a los anteriores utilizar el carácter underscore (_) cuando sea necesario para separar las palabras. Asimismo se recomienda utilizar en el nombre de las variables un sentido completo y humano que refleje el funcionamiento del mismo. Por Ejemplo:

obj_persona

contador

ejeX

controlador

Nomenclatura de Variables

Es recomendable para asegurar la legibilidad del programa y de su entorno usar nombres con prefijos que señalen su clasificación, por ejemplo:

Clasificación	Prefijo	Ejemplo
Global	glb_	glb_contador
Puntero	ptr_	ptr_pilaPeso
Arreglo	arr_	arr_tiempoRecorrido
Instancia de una Clase / Objeto	obj_	obj_funcionario
Formulario	frm_	frm_datosCorrespondencia
Texto	txt_	txt_nombreFuncionario
Select / Combo	cmb_	cmb_profesion
Checkbox	chk_	chk_listaProducto
Radio	rad_	rad_color
Submit	sbm_	sbm_aceptar
Button	btn_	btn_reportePlanilla

Para el tratamiento de codificación en la web es conveniente la siguiente convención:

Clasificación	Prefijo	Ejemplo
Password	psw_	psw_firmaDigital
File	fil_	fil_archivoPrecio
Reset	rst_	rst_limpiar
Hidden	hdn_	hdn_estadoModelo
Método Get	mgt_	mgt_txt_nombreFuncionario
Método Post	mpo_	mpo_cmb_profesion
Método Put	mpu_	mpu_chk_listaProducto

Constantes

Las constantes deben ir en mayúscula, inclusive las constantes de clase. Utilizar el caracter underscore (_) cuando sea necesario para separar las palabras. Por ejemplo:

PATH_MODELO

NOMBRE_SISTEMA

Comentarios

- Los comentarios deben tener el mismo nivel de indentación que el código que comentan.
- Los comentarios deben ser fáciles y sencillos de generar y modificar. Por ejemplo lo expuesto a continuación es opuesto a lo estipulado a esta convención:

```
/******  
*                                           *  
*      Esta es una función que calcula la depreciacion de los activos *  
*      mediante el método de línea recta *  
*                                           *  
*****/
```

- Los comentarios no deben repetir el código ni formularlo de otra manera, sino que deben explicar la intención del mismo.
- En cada módulo, clase, método, función, procedimiento o rutina importante se debe mantener un bloque de comentarios que describa según su necesidad lo siguiente:
 - Su razón, entorno o cometido
 - Los parámetros que acepta, valores o rangos válidos. Precondiciones
 - El resultado que devuelve. Postcondiciones
 - Excepciones que se provocan
 - Efectos secundarios que se provocan, incluyendo cambios en variables globales
- Aquellos métodos o funciones que devuelven valores nulos deben explicar las circunstancias de los mismos (inexistente, sin dato, no aplicable, etc.)
- Las funciones que traten con porcentajes, permiles u otros, deben explicar sus rangos de acción. Por ejemplo: [0-1] , [0-100], [0-1000], ...
- Las funciones matemáticas, físicas u otras que expresen magnitudes deben describir claramente las respectivas unidades de medición

- En caso de desactivar temporalmente fragmentos de código, se debe comentar la razón que expone esta decisión (nuevo requerimiento, afinación, etc.) y cuando se prevé su activación.
- En situaciones que se expresen condiciones particulares con cierta especialidad se recomienda realizar comentarios para ayudar su comprensión; esto generalmente ocurre en los bucles. Por ejemplo:

```
//se busca el caracter delimitador para determinar el parrafo
while(corriente.cadena(posicion) != CARACTER_DELIMITADOR )
{
    .....
}
```

- Se debe aplicar la norma de documentación interna que acompaña al lenguaje de programación utilizado en la creación de un sistema en particular. Por ejemplo phpDocumentor si se utiliza PHP, Javadoc si se usa Java, etc.

Notificación de errores

- Cualquier comunicación de error al usuario debe ser sutil, educado y que además exprese la posible solución. No echar la culpa al usuario.
- En lo posible se debe gestionar un control preventivo de datos para evitar una notificación de errores a posteriori.
- Delimitar claramente una tabla de errores estándar y su posterior tratamiento a estas excepciones y notificarlas de forma clara y concisa, permitiendo al usuario la identificación de errores típicos.

Nombres de Archivos

Los nombres de archivos deben estar en *lowerCamelCase* utilizando el carácter underscore (`_`) cuando sea necesario para separar las palabras. Los nombres de los archivos deben ser significativos o humanizados y que representen el objetivo del archivo. Por ejemplo:

formulario/controlador/formularioServicio.php

libreria/plantilla/datosBasicos.php

ESTILO DE CODIFICACIÓN

Cadenas de caracteres

En lenguajes script como el PHP, las cadenas deben utilizar el caracter de comilla simple (') cuando no se requiera sustitución interna de variables; y cuando exista sustitución debe utilizarse la doble comilla. Por ejemplo:

```
$nombre = 'Daniel Abasto';
```

```
$miNombre = "Mi nombre es $nombre";
```

Concatenación de caracteres

En lenguajes script como el PHP, la concatenación debe realizarse usando el operador punto (.) sin dejar espacios entre los operandos. Por ejemplo:

```
$sql = "select count(*) from ".$tabla;
```

Cuando ocupen varias líneas se debe indentar la concatenación al nivel del operador igual. Por ejemplo:

```
$sql = "select count(*) from ".$tabla.
```

```
    "where campo = 'A'";
```