Sistema de Gestión de Configuración de Software

Documento de Estándares de Programación
Versión 1.0

Historia de Revisión

Historial de revisiones				
Ítem	**Fecha**	**Versión**	**Descripción**	**Equipo**
1	3/9/2022	1.0	Versión Inicial	BDAD

Tabla de Contenidos

Estándares de Programación

1. OBJETIVO

Reglamentar la forma en que se implementará el código fuente del proyecto, pasando, por las variables, controles, clases, métodos, ficheros, archivos y todo aquello que esté implicado en el código,

Mejorar y uniformizar a través de las reglas que se proponen, el estilo de programación que tiene cada programador.

- Los nombres de variables serán mnemotécnicos con lo que se podrá saber el tipo de dato de cada variable con sólo ver el nombre de la variable.
- Los nombres de variables serán sugestivos, de tal forma que se podrá saber el uso y finalidad de dicha variable o función fácilmente con solo ver el nombre de la variable.
- La decisión de poner un nombre a una variable o función será mecánica y automática, puesto que seguirá las reglas definidas por nuestro estándar.
- Permite el uso de herramientas automáticas de verificación de nomenclaturas.

Por tanto, se seguirán dichos patrones para un entendimiento legible del código y para facilitar el mantenimiento del mismo.

2. DECLARACION DE VARIABLES

Se propone que la declaración de las variables, se ajusten al motivo para la que se requieran.

El mnemotécnico definido se establece tomando en consideración principalmente lo siguiente:

- La longitud debe ser lo más recomendable posible. No debe ser tan grande de tal forma que el programador tenga la facilidad de manejo sobre la variable y ni tan corta que no pueda describirse claramente. Para el caso establecemos una longitud máxima de variable de 24 caracteres.

- El tipo de dato al que pertenece la variable.

  Por lo tanto la estructura de la variable es como sigue:

Estructura	**Descripción de la Variable**
LONGITUD. MAX.	← 24 →
FORMATO	*Minúscula la primera parte y luego las siguientes con Mayúsculas*
EJEMPLO	cambioSolicitado

Siendo el nombre que identifica a la variable: **cambioSolicitado**

2.1 Descripción de la Variable.

Nombre que se le asignará a la variable para que se le identifique y deberá de estar asociada al motivo para la cual se le declara.

Ejemplos: idCuenta, tipoEstado.

2.2 Variables de Tipo Arreglo

En el caso de las definiciones de arreglos de elementos se declarará la variable con el prefijo de “lista”, el cual nos dará entender que se trata de una variable del tipo arreglo la cual contendrá de cero a más datos, según el tamaño declarado. El nombre del arreglo debe estar en plural.

Ejemplos: listaTiposDocumento

3. Definición de Controles

Para poder determinar el nombre de un control dentro de cualquier aplicación de tipo visual, se procede a identificar el tipo al cual pertenece y la función que cumple dentro de la aplicación.

3.1 Tipo de datos

Cuatro tipos escalares:

- boolean
- integer
- float (número de punto flotante, también conocido como double)
- String

Cuatro tipos compuestos:

- array
- object
- callable
- iterable

Y finalmente dos tipos especiales:

- resource
- NULL

3.2 Prefijo para el Control

El prefijo del control será determinado mediante tres caracteres que estarán conformados por las consonantes más representativas del control, es así, por ejemplo; el control Button, estará asociado al prefijo btn.

3.3 Nombre descriptivo del Control

Formado por la descripción de la función que lleva a cabo el control, esta debe ser descrita en forma específica y clara.

Tipo de control	Prefijo	Ejemplo
Label	lbl	lblNombre
TextBox	txt	txtApellido
Button	btn	btnLogin
RadioButton	rdo	rdoSeleccion
CheckBox	chk	chkRuta1
DropDownList	cmb	cmbDocumentos

3.4 Declaración de variables, atributos y objetos

1. Se debe declarar una variable por línea.

Título	Descripción
Sintaxis	$[Nombre de la Variable]
Descripción	Todas las variables o atributo tendrán una longitud máxima de 30 caracteres.

	El nombre de la variable puede incluir más de un sustantivo los cuales se escribirán juntos. Si se tuvieran variables que puedan tomar nombres iguales, se le agregará un número asociado (si está dentro de un mismo método será correlativo). En PHP, todas las variables deben empezar con $.
Observaciones	En la declaración de variables o atributos no se deberá utilizar caracteres como: • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, • {, }, [,]. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	**$nombre** Indica una variable que guardará un nombre.

2. Se debe declarar un atributo por línea.

Título	Descripción
Sintaxis	ModificadorAcceso $[Nombre del Atributo]
Descripción	Todas las variables o atributo tendrán una longitud máxima de 30 caracteres. El nombre de la variable puede incluir más de un sustantivo los cuales se escribirán juntos. Si se tuvieran variables que puedan tomar nombres iguales, se le agregará un número asociado (si está dentro de un mismo método será correlativo). En PHP, todos los atributos deben empezar con $. Se debe indicar el modificador de acceso: **public, private o protected.**
Observaciones	En la declaración de variables o atributos no se deberá utilizar caracteres como: • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,]. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	**public $estaConectado** Indica un atributo que guardará el estado Conectado.

3.5 Declaración de clases

Título	Descripción
Sintaxis	**class cls[Nombre de Clase]**
Descripción	El nombre de las clases tendrá una longitud máxima de 30 caracteres y las primera letra del nombre de la clase estará en mayúsculas. No se indica el tipo debido a que no es posible hacerlo en PHP. El nombre de la clase debe estar en **PascalCase**. (Cada palabra siempre empieza con su primera letra mayúscula) Debe tener el prefijo **cls.**
Observaciones	En la declaración de clases no se deberá utilizar caracteres como: • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,]. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	**class clsEmpleado** Indica una clase Empleado

3.6 Declaración de métodos

Título	Descripción
Sintaxis	**ModificadorAcceso function nombreMetodo[(ListaParámetros)]**
Descripción	El nombre del método constará hasta de 25 caracteres. La primera letra de la primera palabra del nombre será escrita en minúscula y las siguientes palabras empezarán con letra mayúscula, es decir, en formato **camelCase.** Se debe indicar el modificador de acceso: **public, private o protected.** Se debe incluir el keyword **function** para los métodos**.**
Observaciones	En la declaración de métodos no se deberá utilizar caracteres como: • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,], _. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	public function calcularSueldo($empleado) Indica un método calcularSueldo que recibe una variable por valor de tipo string al ámbito de la clase

3.7 Declaración de funciones

Título	Descripción
Sintaxis	**function nombreFuncion[(ListaParámetros)]**
Descripción	El nombre del objeto constará hasta de 25 caracteres, no es necesario colocar un nombre que indique la clase a la cual pertenece. El tipo de dato de retorno debe indicarse en el comentario descriptivo de la función, así como, los tipos de dato de los parámetros. Se debe incluir el keyword **function** para los métodos**.**
Observaciones	En la declaración de objetos no se deberá utilizar caracteres como: • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,], _. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	**function sumarNumeros($numero1, $numero2)** Indica una función que suma dos números

4. Clases

El nombre de las clases debe ser auto descriptivo de manera que no se requiera, en lo posible, entrar al código de la función para saber qué es lo que realiza.

Los nombres de las clases deben estar en **StudlyCaps** o **PascalCase** precedidos de las siglas **cls**, la cual debe estar escrita en minúscula seguido del nombre que identifica la clase, la primera letra del nombre debe iniciar con mayúscula.

- Ejemplos: clsCuenta, clsEmpleado, clsFactura

Nota:

- No se hará uso de los caracteres: Espacio en blanco " ", Caracter de subrayado "_".

5. Métodos, Procedimientos y Funciones definidos por el Usuario.

El nombre de las funciones y procedimientos debe ser auto descriptivo de manera que no se requiera, en lo posible, entrar al código de la función para saber qué es lo que realiza.
verbo-Sustantivo

El estándar para nombres de procedimiento es usar un Verbo que describa la acción realizada seguida por un sustantivo (objeto sobre el cual actúa). Se recomienda:

- Usar un nombre que represente una acción y un objeto. El nombre del procedimiento debe indicar qué hace el procedimiento a... o qué hace el procedimiento con....

- El verbo debe estar en infinitivo.

- Ser consistente en el orden de las palabras. Si se va a usar ***verboNombre***, siempre usar ***verboNombre***.

- Ser consistente en los verbos y sustantivos usados. Por ejemplo, si tiene un procedimiento ***asignarNombre***, en vez de ***colocarNombre***.

Ejemplo:

Para la acción **modificar cuenta del cliente** se define:
modificarCuenta
Verbo: modificar
Sustantivo: Cuenta
Cliente: vendría a ser el objeto del que se hace el llamado del método.

Nota:
- No se hará uso de los caracteres: Espacio en blanco " ", Caracter de subrayado "_".
- La nomenclatura de argumentos o parámetros pasados a los procedimientos/funciones, así como para valores devueltos por funciones sigue las mismas convenciones que la **nomenclatura para variables**.

6. Beneficios

- La documentación hace más legible un programa.
- Al documentar bien un programa desde un principio se evita que para cada modificación deba estudiarse profundamente el funcionamiento del programa, redescubriendo todo lo no documentado, con la ventaja adicional de que generalmente quién modifica el programa no es siempre quién lo escribió.
- Facilita la reutilización de módulos y rutinas desde cualquier otro programa o el mismo.
- Ayuda a determinar cuándo debe ser reescrito un código. Si existen problemas para explicar el código con un comentario, probablemente el código esté mal escrito.

7. Conclusiones

- Una buena programación e implementación legible, solo se logra usando y llevando de la mano un buen estándar o patrón de programación.
- Es muy importante que el programador tenga un conocimiento previo del estándar o en su defecto que lea el documento para prever diferencias.
- Al documentar se obtienen dos cosas fundamentales, un documento legible y segundo una buena base para los futuros desarrollos de mantenimiento del código.