**Version 1.0**

**Estándar de Programación de Software**

**MPLEMENTACION DEL ESTANDAR**

1. **OBJETIVOS**

El objetivo del presente documento es definir los estándares de programación con el fin de desarrollar una programación fácil de entender por cualquier programador y contar con una herramienta de fácil documentación de los sistemas de información.

1. **ORGANIZACIÓN DE ARCHIVOS**
   1. **Archivo fuente**

Sufijos de ficheros en Java:

* Código fuente: .java
* Código compilado: .class

Cada archivo fuente debe contener una única clase o interfaz pública. El nombre del archivo tiene qué coincidir con el nombre de la clase. Cuando existan varias clases privadas asociadas funcionalmente a una clase pública, podrán colocarse en el mismo archivo fuente que la clase pública. La clase pública debe estar situada en primer lugar dentro del archivo fuente.

En todo archivo fuente distinguimos las siguientes secciones:

* Comentarios de inicio.
* Sentencia de paquete.
* Sentencias de importación.
* Declaraciones de clases e interfaces.
  1. **Nombres de clases**
* El nombre de cada clase deberá tener la primera letra de cada palabra en mayúscula, deberán estar juntas, sin sub-guiones ni algún carácter que las separe, a excepción del nombre de la carpeta donde se encuentra la clase, en ese caso, si deberá separarse por un sub-guion.
* El nombre de cada clase deberá ser, en la medida de lo posible, simples y descriptivos.
* No se permite el uso de abreviaturas o acrónimos.
* Si la clase cumpliese con alguna funcionalidad especifica es recomendable definirlo en el nombre.

Ejemplo:

Carpeta: Vista

Clase: RegistrarUsuario\_Vista

Carpeta: Modelo

Clase: Usuario\_Modelo

Carpeta: Presentador

Clase: RegistrarUsuario\_Presentador

Carpeta: Interactor

Clase: RegistarUsuario\_Interactor

* 1. **Longitud de línea**

La longitud de línea no debe superar los 140 caracteres por motivos de visualización e impresión.

* 1. **Comentarios**

Distinguimos dos tipos de comentarios: los comentarios de implementación y los de

documentación.

* + 1. **Comentarios de implementación**

Estos comentarios se utilizan para describir el código ("el cómo"), y en ellos se incluye información

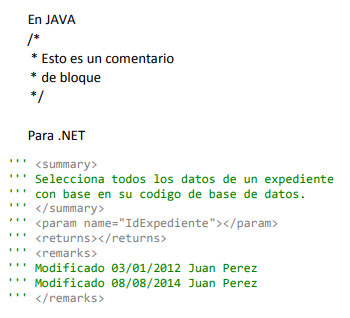
relacionada con la implementación, tales como descripción de la función de variables locales, fases lógicas de ejecución de un método, captura de excepciones, etc.

Distinguimos tres tipos de comentarios de implementación:

* **Comentarios de bloque**:

Permiten la descripción de ficheros, clases, bloques, estructuras de datos y algoritmos.

Ejemplo:



* **Comentarios de línea:**

Son comentarios cortos localizados en una sola línea y tabulados al mismo nivel que el código que

describen. Si ocupa más de una línea se utilizará un comentario de bloque. Deben estar precedidos por una línea en blanco.

Ejemplo:

*/\* Mostrar mensaje del expediente si existe la variable de sesión\*/*

* **Comentario a final de línea**

*Comentario situado al final de una sentencia de código y en la misma línea.*

Ejemplo:

*int contador = 4 + 10; // Inicialización del contador*

*contador++; /\* Incrementamos el contador \*/*

* 1. **Declaraciones**
     1. **Constantes**

Los nombres de las constantes deberán escribirse todo en mayúsculas con palabras separadas por subguion “\_” y todas serán declaradas como private static final.

Ejemplo:

private static final int *PAYPAL\_REQUEST\_CODE*=7171;

* + 1. **Metodos**
* El nombre del método deberá estar en ingles.
* El nombre del método deberá tener la primera letra de cada palabra en mayúscula, a excepción de métodos que empiecen con la palabra “on”.
* El nombre debe ser lo suficientemente descriptivo sin importar la longitud de este.
* No se permite caracteres especiales

***Correcto***:

SaveUser()

onGetUserSuccessful()

***Incorrecto***:

Saveuser()

* + 1. **Variables**
* El nombre de la variable no podrá ser mayor a 25 caracteres.
* El nombre debe ser lo suficientemente descriptivo evitando abreviaturas a excepción de “Btn”, “Txt” y “Lbl”.
* El nombre deberá tener la primera letra de cada palabra en mayúscula y separada por un subguion “\_”.
* El nombre no podrá constar de un solo carácter a excepción de las variables utilizadas en bucles, para esos casos se utilizará i, j, k, l (siempre en ese orden).

***Correcto:***

Id\_Cupon

***Incorrecto:***

IC, IDc, ID\_Cupon

* + 1. **Una declaración por línea**

Se recomienda el uso de una declaración por línea, promoviendo así el uso de comentarios.

Ejemplo:

int Id\_Unidad;

EditText TxtCorreo\_Electronico ;

* + 1. **Inicialización**

Inicializar cada variable en su declaración a menos que su valor inicial dependa de algún cálculo.

int Id\_Unidad = 1;

String[] Funciones = { "Administración", "Intervención", "Gestión" };

* + 1. **Localización.**

En el comienzo de los bloques. La única excepción es el bucle *for*. No se deben declarar variables con el mismo nombre que otras en un método.

Ejemplo:

for (int i=0; contador<10; i++)

{

...

}

Se debe evitar el uso de declaraciones que oculten a otras declaraciones de ámbito superior.

int Contador = 0; // Inicio del método

public void Method()

{

if (condicion)

{

int Contador = 2; // ¡¡ EVITAR !!

...

}

...

}

* + 1. **Declaración de clases / interfaces**

Durante el desarrollo de clases / interfaces se deben seguir las siguientes reglas de formateo:

* No incluir ningún espacio entre el nombre del método y el paréntesis inicial del listado
* de parámetros.
* El carácter inicio de bloque ("{") debe aparecer al final de la línea que contiene la sentencia de declaración.
* El carácter fin de bloque ("}") se sitúa en una nueva línea tabulada al mismo nivel que

su correspondiente sentencia de inicio de bloque, excepto cuando la sentencia sea nula, en tal caso se situará detrás de "{".

* Los métodos se separarán entre sí mediante una línea en blanco.

Ejemplo:

public classe ClaseEjemplo extends Object {

int Variable1 = 0;

int Variable2 = 1;

public ClaseEjemplo() {

variable1 = 0;

variable2 = 1;

}

...

}

* 1. **Sentencias y Bucles**
* Todo bloque de sentencia deberá ir entre llaves y en una nueva línea de código.
* Toda condición va dentro de paréntesis.

***Correcto:***

For (i=0; i<10; i++)

{

}

***Incorrecto***:

For (i=0; i<5; i++){

}

* No es necesario que cada if deba tener su respectivo else, solo si no habrá código dentro de este.
* Siempre se debe usar identanción en cada bloque de código.

***Correcto***:

For (i=0; i<10; i++)

{

If (I == 2)

{

…

}

else

{

//Solo si hay código dentro

}

}

* + 1. **Sentencias de retorno**

No se pondrá la expresión de retorno entre paréntesis a menos que con ello se gane en claridad

* 1. **Espacios en blanco**

Las líneas y espacios en blanco mejoran la legibilidad del código permitiendo identificar las secciones de código relacionadas lógicamente.

Se utilizarán espacios en blanco en los siguientes casos:

Entre una palabra clave y un paréntesis. Esto permite que se distingan las llamadas a métodos de

las palabras clave. Por ejemplo:

while (true)

{

...

}

Tras cada coma en un listado de argumentos. Por ejemplo:

objeto.unMetodo(a, b, c);

Para separar un operador binario de sus operandos, excepto en el caso del operador ("."). Nunca

se utilizarán espacios entre los operadores unarios (p.e., "++" o "--") y sus operandos. Por ejemplo:

a += b + c;

a = (a + b) / (c + d);

contador++;

Para separar las expresiones incluidas en la sentencia "for". Por ejemplo:

for (expresion1; expresion2; expresion3)

Al realizar el moldeo o "casting" de clases. Ejemplo:

Unidad unidad = (Unidad) objeto;

* 1. **Prácticas de programación**
     1. **Visibilidad de atributos de instancia y de clase**

Los atributos de instancia y de clase serán siempre privados, excepto cuando tengan que ser visibles en subclases herederas, en tales casos serán declarados como protegidos.

El acceso a los atributos de una clase se realizará por medio de los métodos "get" y "set" correspondientes (propiedades), incluso cuando el acceso a dichos atributos se realice en los métodos miembros de la clase.

Ejemplo:

public class Unidad {

private int id;

private String nombre;

...

public void UpdateUnit(Unidad unidad) {

this.setId(unidad.getId());

this.setNombre(unidad.getNombre());

}

...

}

* + 1. **Referencias a miembros de una clase**

Evitar el uso de objetos para acceder a los miembros de una clase (atributos y métodos estáticos).

Utilizaremos en su lugar el nombre de la clase.

Por ejemplo:

metodoUtilidad(); // Acceso desde la propia clase estática

ClaseUtilidad.metodoUtilidad(); // Acceso común desde cualquier clase.

* + 1. **Asignación sobre variables**

Se deben evitar las asignaciones de un mismo valor sobre múltiples variables en una misma sentencia, ya que dichas sentencias suelen ser difíciles de leer.

Ejemplo:

int a = b = c = 2; // Evitar

No utilizar el operador de asignación en aquellos lugares donde sea susceptible de confusión con

el operador de igualdad.

Por ejemplo:

// INCORRECTO

if ((c = d++) == 0)

{

}

// CORRECTO

c = d++;

if (c == 0)

{

}

No utilizar asignaciones embebidas o anidadas. Ejemplo:

c = (c = 3) + 4 + d; // Evitar

Debería escribirse

c = 3;

c = c + 4 + d;

1. **DOCUMENTACION**

* + 1. **Sobre la Documentación**

Durante el desarrollo del sistema cada fase terminada deberá ser documentada, según el estándar de documentación vigente. La responsabilidad de la documentación de cada sistema de información recaerá en el jefe de cada proyecto de software.

Se deberá indicar en la documentación el código identificador asignado para cada tabla que conforma el sistema desarrollado.

Todo sistema que se encuentre en producción deberá tener un registro de bitácora, donde se detallen las modificaciones y/o adecuaciones que se realicen sobre el mismo.