**Estándares de Programación JAVA**

**1. Introducción**

El objetivo es el establecimiento de los diferentes estándares que pueden ser empleador durante el desarrollo de software sobre el lenguaje JAVA.

Las ventajas que podemos tener para el uso de estos estándares son:

* Facilidad en Mantenimiento de la aplicación.
* Permite el fácil entendimiento para cualquier persona.
* Mejora de la legibilidad del código.

**2. Ficheros o Archivos**

**2.1 Organización**

En el desarrollo de la aplicación, las diferentes clases en Java se las debe agrupar por paquetes, por lo que se les debe organizar de una manera jerárquica seguidos por un punto como separador.

Así dentro del paquete principal se deben organizar los diferentes subpaquetes dependiendo de las diferentes funciones que vaya a cumplir dicha aplicación. Por ejemplo, desarrollamos un web Service de acceso a datos debería tener la siguiente estructura: imfe.webservice.data, donde **imfe** es el nombre de la aplicación, **webservice** el paquete principal y **data**subpaquete.

Cada clase de java debe contar de lineas separadas y comentarios que identifiquen cada sección de la misma.

Deben evitarse archivos de gran tamaño que no contengan mas de 1000 lineas, ya que puede ocasionar un mal entendimiento albergando gran cantidad de métodos.

**2.2 Archivo o Fichero fuente Java (.java)**

Cada archivo java debe contener una única clase o interfaz. El nombre del archivo tiene que ser el mismo que el nombre de la clase. En el caso que existan varias clases privadas que estén vinculadas a la clase publica, se debería colocar en el mismo archivo que la clase publica por lo que esta clase publica debe estar ubicada en primer lugar antes que las clases privadas.

Todo archivo Java debe tener las siguientes secciones:

* Comentarios de Inicio
* Sentencia de paquete
* Sentencia de importación
* Declaraciones de clases e interfaces

**2.2.1 Comentarios de Inicio**

Todo archivo fuente debe comenzar con un comentario que incluya:

* Nombre de la clase
* Información de la version
* Fecha
* Copyright

En el caso del *copyright*, esta debe indicar:

* Propiedad legal del código
* Ambito de distribución,
* Uso para el que fue desarrollado
* Modificación

Adicionalmente podrían incluirse opcionalmente:

* Comentarios sobre los cambios efectuados en dicho archivo
* Historial de cambios realizados en el archivo

*Ejemplo de Comentario Inicial*

/\*

\* @(#)JceSecurity.java 1.50 04/04/14

\*

\* Copyright 2004 Sun Microsystems, Inc. All rights reserved.

\* SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.

\*/

/\*\*

\* This class instantiates implementations of JCE engine classes from

\* providers registered with the java.security.Security object.

\*

\* @author Jan Luehe

\* @author Sharon Liu

\* @version 1.50, 04/14/04

\* @since 1.4

\*/

**2.2.2 Sentencia de paquetes**

La primera linea que debe contener un archivo fuera del comentario inicial, son los paquetes al que pertenece, clases incluidas en el archivo.

*Por ejemplo:*

package javax.crypto;

**2.2.3 Sentencia de importación**

Después de la declaración del paquete, se debe incluir las sentencias de importación de los paquetes necesarios para el desarrollo del programa, el cual debería seguir el siguiente lineamiento:

* Paquetes JDK ([Java Development Kit](http://es.wikipedia.org/wiki/Java_Development_Kit))
* Paquetes de utilidades que no pertenezcan al JDK como frameworks o proyectos opensource.
* Paquetes desarrollados para la aplicación.

Lo que se recomienda es minimizar el uso de importacion completa de paquete, es decir del tipo *"package.\*"*, ya que dificulta la comprension de las dependencias del cual estan siendo usadas en la aplicación.

*Ejemplo:*

import java.io.\*; // BufferedReader, PrintWriter, FileInputStream, File

import java.util.ArrayList;

import org.apache.log4j.Logger;

import org.apache.lucene.analysis.Analyzer;

import provincia.organismo.corporativas.atlas.vo.AgendaVO;

**2.2.4 Declaraciones de clases e interfaces**

La siguiente tabla nos mostrara los elementos que componen la declaración de una clases asi como el orden al que deben ser usados:

|  |  |
| --- | --- |
| **Elementos de declaración de una clase / interfaz** | **Descripción** |
| Comentario de documentación de la clase/interfaz /\*\* ... \*/ | Permite describir la clase/interfaz desarrollada. Necesario para generar la documentación de la api mediante javadoc. |
| Sentencia class / interface |  |
| Comentario de implementación de la clase/interfaz, si es necesario /\* ... \*/ | Este comentario incluye cualquier información que no pueda incluirse en el comentario de documentación de la clase/interfaz. |
| Variables de clase (estáticas) | En primer lugar las variables de clase públicas (public), después las protegidas (protected), posteriormente las de nivel de paquete (sin modificador), y por último las privadas (private). |
| Variables de instancia | Primero las públicas (public), después las protegidas (protected), luego las de nivel de paquete (sin modificador), y finalmente las privadas (private). |
| Constructores |  |
| Métodos | Deben agruparse por funcionalidad en lugar de agruparse por ámbito o accesibilidad. Por ejemplo, un método privado puede estar situado entre dos métodos públicos. El objetivo es desarrollar código fácil de leer y comprender. |

**3. Sangria**

La norma general es establecer cuatro caracteres como unidad de sangria. En los diferentes IDE([Entorno de Desarrollo Integrado](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCoQFjAA&url=http%3A%2F%2Fes.wikipedia.org%2Fwiki%2FEntorno_de_desarrollo_integrado&ei=K5uIUvmgBOrF7AbnxIC4Bg&usg=AFQjCNH4d_y0Ju95hXzCWY9Gr2vTY7kLdA&sig2=AwfTfiPK5uFHEg46Csj5aw&bvm=bv.56643336,d.d2k)) mas conocidos como Eclipse o NetBeans, estos incluyen la facilidad de darle formato al código Java.

**3.1 Longitud de línea**

No debe superar mas de 80 caracteres por motivos de visualización e impresión.

**3.2 División de líneas**

En ocasiones una expresión puede ocupar mas de una línea, por lo que se deberá romper dicha expresión de acuerdo a los siguientes criterios:

* Tras una coma
* Antes de un operador
* Rupturas de nivel superior a las de nivel inferior
* Alinear la nueva línea con el inicio de la expresión al mismo nivel que la línea anterior

*Ejemplo:*

unMetodo(expresionLarga1, expresionLarga 2, expresionLarga 3,

expresionLarga 4, expresionLarga 5);

if ((condicion1 && condicion2)

|| (condicion3 && condicion4)

||!(condicion5 && condicion6)) {

unMetodo();

}

**4. Comentarios**

**4.1 Comentarios de Implementación**

Se usan para describir el código, el como, en donde se incluye información relacionada con la implementación, como la descripción de la funciones de las variables locales, fases lógicas de ejecución del método, captura de excepciones, etc.

**4.1.1 Comentarios de bloque**

Descripción de archivos, clases, bloques, estructuras de datos y algoritmos.

*Ejemplo*

/\*

\* Esto es un comentario

\* de bloque

\*/

**4.1.1 Comentarios de línea**

Son comentarios cortos, que están localizados en una sola línea y tabulados al mismo nivel que el código que lo describe.

*Ejemplo:*

/\* Esto es un comentario de línea \*/

// Esto es otro comentario de línea

**4.1.1 Comentarios fuera de línea**

Es el comentario ubicado al final de la sentencia del mismo código o línea de código

*Ejemplo:*

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

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

**4.2 Comentarios de Documentación**

También denominados comentarios Javadoc, en el que utilizan para describir la especificación del código, desde un punto de vista independiente a la implementación. Para ser consultada por desarrolladores que no tengan acceso al código fuente.

**5. Declaraciones**

**5.1 Declaración por línea**

Es aconsejable ya que se promueve el uso de comentarios

*Ejemplo:*

int idUnidad; // Identificador de la unidad organizativa

String[] funciones; // Funciones de la unidad

**5.2 Inicialización**

Toda variable local debe ser inicializada en el momento de su declaración, excepto que el valor que contenga inicialmente dependa de algún otro valor ya calculado previamente.

*Ejemplo:*

int idUnidad = 1;

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

**5.3 Localización**

**5.3.1 Variables Locales**

Las declaraciones deben estar situadas al principio de cada bloque principal y nunca en el momento de su uso.

*Ejemplo:*

public void unMetodo() {

int contador = 0; // inicio del método

...

}

**5.3.2 Variables Globales**

Estas variables deben estar ubicadas al inicio de la clase principal

*Ejemplo:*

public class unaClase{

int contador = 0; // Inicio de la variable Global

public void unMetodo() {

if (condicion) {

int contador = 2; // ¡¡ EVITAR !!

...

}

...

}

}

**5.3.3 Excepciones**

Cuando se usan los indices de bucle **for**ya que en Java se las puede incluir dentro de la misma sentencia

*Ejemplo:*

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

...

}

**5.4 Declaración de clases o interfaces**

Deben seguir el siguiente formato:

* No incluir espacios entre el nombre del método y los parentesis
* El carácter de inicio del bloque es { y debe aparecer al final de la sentencia de declaración } debidamente tabulada al mismo nivel correspondiente
* Cada método se separa de otro por medio de una linea en blanco

*Ejemplo:*

public classe ClaseEjemplo extends Object {

int variable1;

int variable2;

public ClaseEjemplo() {

variable1 = 0;

variable2 = 1;

}

...

}

**6. Sentencias**

Cada línea debe tener como máximo una sentencia

Ejemplo:

int contador++;

int variable--;

Toda sentencia de un bloque debe estar contendida entre llaves {...}, aunque sea de una única sentencia.

Ejemplo:

if (condicion) {

variable++;

}

La sentencia try/catch tiene que tener este formato:

try {

sentencias;

} catch (ClaseException e) {

sentencias;

}

Para liberar los recursos de este bloque se usa "*finally*"

try {

sentencias;

} catch (ClaseException e) {

sentencias;

} finally {

sentencias;

}

**7. Espacios en Blanco**

Permiten una mejor legibilidad del código para poder identificar las secciones de código relacionadas. Y se utilizara los espacios en blanco en los siguientes casos:

* Entre una palabra claver y un parentesis, para poder distinguir las llamadas a métodos de las palabras claves.

while (true) {

...

}

* Tras cada coma en un listado de argumentos

objeto.unMetodo(a, b, c);

* Separar un operador binario de sus operandos

a += b + c;

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

contador++;

* Separa expresiones incluidas en la sentencia *for*

for (expresion1; expresion2; expresion3)

* Al realizar *casting* de clases

Unidad unidad = (Unidad) objeto;