## Estándar de Programación

## CODIGO DE JAVA

## Indentación

Para la indentación de nuestra codificación se deben emplear cuatro espacios como unidad de indentación, ésta puede ser configurada en eclipse.

**Longitud de la línea**

Evitar las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas terminales y herramientas.

**Salto de líneas**

Cuando una expresión no entre en una línea, se debe hacer el salto de línea de acuerdo a los siguientes principios:

* Después de una coma.
* Antes de un operador.
* Preferir saltos de línea de alto nivel (más a la derecha que el "padre") que de bajo nivel (más a la izquierda que el "padre").
* Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la anterior.

Si las reglas anteriores llevan a código confuso o a código que se aglutina en el margen derecho, indentar justo 8 espacios en su lugar.

Ejemplos de cómo hacer los saltos de línea en la llamada a un método:

unMetodo(expresionLarga1, expresionLarga2, expresionLarga3,

expresionLarga4, expresionLarga5);

var = unMetodo1(expresionLarga1,

unMetodo2(expresionLarga2,

expresionLarga3));

Ahora dos ejemplos de saltos de líneas en expresiones aritméticas. Se recomienda utilizar el primer ejemplo, ya que el salto de línea ocurre fuera de la expresión que encierra los paréntesis.

nombreLargo1 = nombreLargo2 \* (nombreLargo3 + nombreLargo4

- nombreLargo5) + 4 \* nombreLargo6; // PREFERIDA

nombreLargo1 = nombreLargo2 \* (nombreLargo3 + nombreLargo4

- nombreLargo) + 4 \* nombreLargo6; // EVITAR

Al momento de realizar un salto de línea en una sentencia, por ejemplo *if*, se recomienda indentar con 8 espacios en vez de 4, de manera que se vea más clara la estructura, por ejemplo:

//NO USAR ESTA INDENTACION

if ((condicion1 && condicion2)

|| (condicion3 && condicion4)

||!(condicion5 && condicion6)) { //MALOS SALTOS

hacerAlgo(); //HACEN ESTA LINEA FACIL DE OLVIDAR

}

//USAR LA SIGUIENTE INDENTACION

if ((condicion1 && condicion2)

|| (condicion3 && condicion4)

||!(condicion5 && condicion6)) {

hacerAlgo();

}

//O USAR ESTA

if ((condicion1 && condicion2) || (condicion3 && condicion4)

||!(condicion5 && condicion6)) {

hacerAlgo();

}

**Declaraciones**

**Cantidad por línea**

Se recomienda una declaración por línea, ya que facilita los comentarios. Se acepta la declaración agrupada de variables si es que aquellas están relacionadas en algún procedimiento o estructura. Ejemplos:

int nivel; // nivel de indentación

int tam; // tamaño de la tabla

int ancho, alto; //dimensiones de la matriz

Nota: Los ejemplos anteriores usan un espacio entre el tipo y el comentario. Una alternativa aceptable es usar tabuladores, con lo que se obtiene más orden, por ejemplo:

int level; // nivel de indentación

int size; // tamaño de la tabla

Object currentEntry; // entrada de la tabla seleccionada actualmente

**Inicialización**

Intentar inicializar las variables locales donde se declaran. La única razón para no inicializar una variable donde se declara es si el valor inicial depende de algunos cálculos que deben ocurrir. Con la inicialización evitamos posibles fallas en el uso de éstas

**Ubicación**

Orientar las declaraciones solo al principio de los bloques (un bloque es cualquier código encerrado por llaves "{" y "}".) No esperar al primer uso para declararlas; puede confundir y limitar la portabilidad del código dentro de su ámbito de visibilidad.

void myMethod() {

int int1 = 0; // comienzo del bloque del método

if (condition) {

int int2 = 0; // comienzo del bloque del "if"

...

}

}

La excepción de la regla son los indices de bucles *for*, que en Java se pueden declarar dentro de la sentencia:

for (int i = 0; i < N; i++) { ... }

Evitar las declaraciones locales que ocultan otras de niveles superiores, por ejemplo, no declarar la misma variable en un bloque interno:

int cuenta;

...

miMetodo() {

if (condicion) {

int cuenta = 0; // EVITAR!

...

}

...

}

**Declaraciones de class e interfaces**

Al codificar clases e interfaces de Java, se siguen las siguientes reglas de formato:

* Ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros.
* La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración y no en la línea siguiente.
* La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{"

class Ejemplo extends Object {

int ivar1;

int ivar2;

Ejemplo(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int metodoVacio() {}

...

}

**Sentencias**

**Sentencias simples**

Cada línea debe contener como mucho una sentencia. Ejemplo:

argv++; // Correcto

argc--; // Correcto

argv++; argc--; // EVITAR!

**Sentencias compuestas**

Las sentencias compuestas son aquellas que contienen listas de procedimientos encerrados entre llaves " { sentencias }". Ellas siguen las siguientes reglas:

* Las sentencias encerradas deben indentarse un nivel más que la sentencia compuesta.
* La llave de apertura se debe poner al final de la línea que comienza la sentencia la llave de cierre debe empezar una nueva línea después de la última instrucción y ser indentada al mismo nivel que el pincipio.
* Las llaves se usan en todas las sentencias, incluso las simples, cuando forman parte de una estructura de control, como en las sentencias *if-else* o *for*. Esto hace más sencillo añadir instrucciones sin producir errores accidentalmente por olvidar las llaves.

**Sentencias de retorno**

Una sentencia *return* con un valor no debe usar paréntesis. Ejemplos:

*return 0;*

*return arreglo.size();*

**Sentencias if, if-else, if else-if else**

La clase de sentencias if-else debe tener la siguiente forma:

if (condicion) {

sentencias;

}

if (condicion) {

sentencias;

}

else {

sentencias;

}

if (condicion) {

sentencia;

*}*

else {

if (condicion) {

sentencia;

}

else{

sentencia;

}

}

Nota: Las sentencias *if* como *else* usan siempre llaves {}. Evitar la siguiente forma, propensa a errores:

if (condicion) //EVITAR OMITIR LAS LLAVES COMO SE DIJO ANTERIORMENTE

sentencia;

**Sentencias for**

Una sentencia for debe tener la siguiente forma:

for (inicializacion; condicion; actualizacion) {

sentencias;

}

Una sentencia for vacía (una en la que todo el trabajo se hace en las clausulas de

inicialización, condicion, y actualizacion) debe tener la siguiente forma:

for (inicializacion; condicion; actualizacion);

Al usar el operador punto y coma en la clausula de inicialización o actualización de una sentencia *for*, evitar la complejidad de usar más de tres variables, éstas se deben manejar antes o después del ciclo según sea necesario.

**Sentencias while**

Una sentencia *while* debe tener la siguiente forma:

while (condicion) {

sentencias;

}

Una sentencia *while* vacía debe tener la siguiente forma:

while (condicion);

**Sentencias do-while**

Una sentencia *do-while* debe tener la siguiente forma:

do {

sentencias;

} while (condicion);

**Sentencias switch**

Una sentencia *switch* debe tener la siguiente forma:

switch (condicion) {

case ABC:

sentencias;

/\* este caso se propaga \*/

case DEF:

sentencias;

break;

case XYZ:

sentencias;

break;

default:

sentencias;

break;

}

Cada vez que un caso se propaga (no incluye la sentencia break), añadir un comentario donde la sentencia break se encontraría normalmente. Esto se muestra en el ejemplo anterior con el comentario */\* este caso se propaga \*/.*

Cada sentencia switch debe incluir un caso por defecto. El break en el caso por defecto es redundante, pero prevee que se propague por error si luego se añade otro caso. Además evitar dejar casos sin su correspondiente break.

**Sentencias try-catch**

Una sentencia try-catch debe tener la siguiente forma:

try {

sentencias;

} catch (ExceptionClass e) {

sentencias;

}

Una sentencia try-catch puede ir seguida de un finally, cuya ejecución se ejecutará independientemente de que el bloque try se haya completado con éxito o no.

try {

sentencias;

} catch (ExceptionClass e) {

sentencias;

} finally {

sentencias;

}

## Convenciones de nombres

**Clases**

Los nombres de las clases deben ser **sustantivos**, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases simples y descriptivas. Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo).

Ejemplos:

class **C**liente;

class **I**magen**A**nimal;

## Métodos

Los métodos deben ser **verbos**, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

Ejemplos:

**e**jecutar();

**e**jecutar**R**apido();

**e**jecutar**M**uy**R**apido();

1. Nombres de métodos para Administración:
2. Registrar Objeto: **add**Objeto.
3. Editar Objeto: **edit**Objeto.
4. Eliminar Objeto: **del**Objeto.
5. Retornar Objeto: **get**Objeto.
6. Guardar Objeto: **set**Objeto.

Donde “Objeto” es el nombre de la clase como: Usuario, Cliente, Mascota, etc.

Para el caso de los objetos como Usuario, Mascota, Cliente, etc. sus métodos para interacción de atributos son:

1. Retornar Atributo: **get**Atributo
2. Guardar Atributo: **set**Atributo

Donde “Atributo” es el nombre completo del atributo pero con mayúscula por ejemplo, para retornar atributo “nombre” de la Clase Cliente el método se llamaría getNombre.

1. Parámetros de Métodos

El parámetro de entrada del método es el nombre completo en minúscula del atributo al que se refiere, Por ejemplo:

getUsuario(Usuario **usuario**)

setNombre(String **nombre**)

## Variables

Excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas (como el caso de los métodos). Los nombres de las variables deben ser cortos pero con significado. Los nombres de variables de un solo carácter se deben evitar, excepto para variables índices temporales. Nombres comunes para variables temporales son i, j, k, m, y n para enteros (muy utilizados como contadores en ciclos for y while); c, d, y e para caracteres.

## Prefijos en Variables Globales

En el caso de que se utilicen variables que son únicamente de un objeto, y que son utilizadas en alguna clase de interfaz u otra en que intervienen otras clases u objetos, se recomienda utilizar un prefijo, que debe ser el nombre de la clase u objeto, seguido por un ‘\_’ y luego el nombre de la variable.

Ejemplos:

String mascota\_nombre;

String cliente\_nombre;

String mascota\_edad;

String cliente\_edad;

## Constantes

Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un subguión ("\_").

Ejemplos:

static final int ANCHURA\_MINIMA = 4;

static final int ANCHURA\_MAXIMA = 99;

static final int NUMERO\_DE\_RAZAS = 100;

## Hábitos

## Proporcionando acceso a variables de instancia y de clase

Para que la parte lógica sea netamente orientada a objetos y el sistema sea seguro solo se permitirán atributos privados en una clase, por ende se restringe absolutamente la utilización de atributos públicos. Para poder acceder a estos se utilizaran métodos. Ya que a menudo las variables de instancia no necesitan ser asignadas/consultadas explícitamente, a menudo esto sucede como efecto lateral de llamadas a métodos.

## Referencias a variables y métodos de clase

Evitar usar un objeto para acceder a una variable o método de clase (static). Usar el nombre de la clase en su lugar.

Ejemplos:

CORRECTO

metodoDeClase();

UnaClase.metodoDeClase();

EVITAR

unObjeto.metodoDeClase();

## Asignaciones de variables

Evitar asignar el mismo valor a varias variables en la misma sentencia. Es difícil de leer.

Ejemplo:

EVITAR

fooBar.fChar = barFoo.lchar = 'c';

No usar asignaciones caladas como un intento de mejorar el rendimiento en tiempo de ejecución. Ese es el trabajo del compilador.

Ejemplos:

EVITAR

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

CORRECTO

a = b + c;

d = a + r;

## Paréntesis

En general es una buena idea usar paréntesis en expresiones que implican distintos operadores para evitar problemas con el orden de precedencia de los operadores. Incluso si parece claro el orden de precedencia de los operadores, podría no ser así para otros, no se debe asumir que otros programadores conozcan el orden de precedencia.

Ejemplos:

EVITAR

if (a == b && c == d)

CORRECTO

if ((a == b) && (c == d))

## Variables temporales

Evitar variables temporales que no son posteriormente utilizadas o que son utilizadas una sola vez, esto para simplificar código, obviamente esto es aplicable si no complica su interpretación.

Ejemplos:

EVITAR

double basePrice = anOrder.basePrice();

return(basePrice > 100);

CORRECTO

return(anOrder.basePrice() > 1000);

En el ejemplo se puede ver que basePice es únicamente utilizada para retornar un valor en un función, pero esto se puede realizar de manera más directa y retornando anOrder.basePrice();

En caso de que una expresión de condición sea muy larga y compleja es mejor utilizar la forma anteriormente descrita como errónea.

Ejemplos:

EVITAR

if( ( platform.toUpperCase().indexOf(“MAC”) > -1 ) &&

( browser.toUpperCase().indexOf(“IE”) > -1) &&

wasInitialized() && resize > 0 ){

// do something

}

CORRECTO

final boolean isMacOs = plataform.toUpperCase().indexOf(“MAC”) > -1;

final boolean isIEBrowser = plataform.toUpperCase().indexOf(“IE”) > -1;

final boolean wasResized = resize > 0;

if (isMacOs && isIEBowser && wasInitialized() && wasResized){

// do something

}

## Sobre la administración de paquetes

En nuestro proyecto BonatiServer dentro del conjunto de códigos fuentes se encuentan distintos paquetes que organizan la estructura de nuestro sistema, los que podemos clasificar dentro de 3 grupos:

* **Bd**: paquete que contiene la clase DBConnectionManager.java, la que se encarga de conectar la capa 2 con la 3.
* **Administracion**: Contiene clases que gestionan los datos de la veterinaria. De manera que se trabaje más cómodamente con el SVN, este paquete tiene cada una de las clases por separadas, por ejemplo:
  + **addUsuario.java**
  + **getUsuarios.java**
  + **…**
* **AdministracionDB:** Contiene la equivalencia de las clases anteriores, pero éstas se conectan con la base de datos, deben seguir la siguiente forma:
  + **addUsuarioBD.java**
* **TransferObjects:** Clases de nuestro sistema que modelan las entidades, estas son equivalentes con las que existen en Flex, ejemplos:
  + **Usuario.java**
  + **Mascota.java**
  + **…**

## ESTANDAR CODIGO DE ACTION SCRIPT

## Indentación

Para la indentación de nuestra codificación se deben emplear cuatro espacios como unidad de indentación, ésta puede ser configurada en Flex.

**Longitud de la línea**

Evitar las líneas de más de 80 caracteres, ya que no son manejadas bien por muchas terminales y herramientas.

**Salto de líneas**

Cuando una expresión no entre en una línea, se debe hacer el salto de línea de acuerdo a los siguientes principios:

* Después de una coma.
* Antes de un operador.
* Preferir saltos de línea de alto nivel (más a la derecha que el "padre") que de bajo nivel (más a la izquierda que el "padre").
* Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la anterior.

Si las reglas anteriores llevan a código confuso o a código que se aglutina en el margen derecho, indentar justo 8 espacios en su lugar.

Ejemplos de cómo hacer los saltos de línea en la llamada a un método:

unaFuncion(expresionLarga1, expresionLarga2, expresionLarga3,

expresionLarga4, expresionLarga5);

var variable:int = unMetodo1(expresionLarga1,

unMetodo2(expresionLarga2,

expresionLarga3));

Ahora dos ejemplos de saltos de líneas en expresiones aritméticas. Se recomienda utilizar el primer ejemplo, ya que el salto de línea ocurre fuera de la expresión que encierra los paréntesis.

CORRECTO:

nombreLargo1 = nombreLargo2 \* (nombreLargo3 + nombreLargo4

- nombreLargo5) + 4 \* nombreLargo6;

EVITAR:

nombreLargo1 = nombreLargo2 \* (nombreLargo3 + nombreLargo4

- nombreLargo) + 4 \* nombreLargo6;

Al momento de realizar un salto de línea en una sentencia, por ejemplo *if*, se recomienda indentar con 8 espacios en vez de 4, de manera que se vea más clara la estructura, por ejemplo:

EVITAR:

if ((condicion1 && condicion2)

|| (condicion3 && condicion4)

||!(condicion5 && condicion6)) { //MALOS SALTOS

hacerAlgo(); //HACEN ESTA LINEA FACIL DE OLVIDAR

}

CORRECTO:

if ((condicion1 && condicion2)

|| (condicion3 && condicion4)

||!(condicion5 && condicion6)) {

hacerAlgo();

}

CORRECTO:

if ((condicion1 && condicion2) || (condicion3 && condicion4)

||!(condicion5 && condicion6)) {

hacerAlgo();

}

**Declaraciones**

**Cantidad por línea**

Se recomienda una declaración por línea, ya que facilita los comentarios. Se acepta la declaración agrupada de variables si es que aquellas están relacionadas en algún procedimiento o estructura. Ejemplos:

var nivel:int; // nivel de indentación

var tam:int; // tamaño de la tabla

Nota: Los ejemplos anteriores usan un espacio entre el tipo y el comentario. Una alternativa aceptable es usar tabuladores, con lo que se obtiene más orden, por ejemplo:

var nivel:int; // nivel de indentación

var size:int; // tamaño de la tabla

**Inicialización**

Intentar inicializar las variables locales donde se declaran. La única razón para no inicializar una variable donde se declara es si el valor inicial depende de algunos cálculos que deben ocurrir. Con la inicialización evitamos posibles fallas en el uso de éstas.

**Sentencias**

**Sentencias simples**

Cada línea debe contener a lo más una sentencia, de igual forma que en el estándar de java.

**Sentencias compuestas**

Se deben cumplir las siguientes reglas:

* Las sentencias encerradas deben indentarse un nivel más que la sentencia compuesta.
* La llave de apertura se debe poner al final de la línea que comienza la sentencia la llave de cierre debe empezar una nueva línea después de la última instrucción y ser indentada al mismo nivel que el pincipio.
* Las llaves se usan en todas las sentencias, incluso las simples, cuando forman parte de una estructura de control, como en las sentencias *if-else* o *for*. Esto hace más sencillo añadir instrucciones sin producir errores accidentalmente por olvidar las llaves.

**Sentencias de retorno**

Una sentencia *return* con un valor no debe usar paréntesis.

Ejemplos:

return 0;

return arreglo.size();

**Sentencias if, if-else, if else-if else**

La clase de sentencias if-else debe tener la siguiente forma:

if (condicion) {

sentencias;

}

if (condicion) {

sentencias;

}

else {

sentencias;

}

if (condicion) {

sentencia;

*}*

else {

if (condicion) {

sentencia;

}

else{

sentencia;

}

}

Se debe evitar no usar llaves

**Sentencias while**

Una sentencia *while* debe tener la siguiente forma:

while (condicion) {

sentencias;

}

Una sentencia *while* vacía debe tener la siguiente forma:

while (condicion);

**Sentencias do-while**

Una sentencia *do-while* debe tener la siguiente forma:

do {

sentencias;

} while (condicion);

## Convenciones de nombres

**Clases**

Al igual que en el caso de Java los nombres de las clases deben ser **sustantivos** y cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas, cabe señalar que los nombres de las clases en esta capa deben ser idénticos a la segunda capa (Java), mas adelante se explicara.

Ejemplos:

**C**liente.as;

**I**magen**A**nimal.as;

## Métodos

Los métodos en actionscript también deben ser **verbos** y cuando sean compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula.

Ejemplos:

**public** **function** ejecutar():**void**

**public** **function** ejecutarRapido():**void**

**public** **function** ejecutarMuyRapido():**void**

Los nombres de las clases en actionscript deben ser los mismos con los que se comunican con java.

Por ejemplo:

Mascota.java -> Mascota.as

MascotaService.java -> MascotaService.as

Los Prefijos de los métodos de la clase ObjetoService son:

1. Registrar Objeto: **add**Objeto.
2. Editar Objeto: **edit**Objeto.
3. Eliminar Objeto: **del**Objeto.
4. Retornar Objeto: **get**Objeto.
5. Guardar Objeto: **set**Objeto.

En esta capa los objetos tienen atributos públicos, por ejemplo:

**public** **var** rutCliente:String;

**Parámetros de Métodos**

El parámetro de entrada de los métodos para las clases ObjetoService, es el nombre completo en minúscula del atributo al que se refiere, Por ejemplo:

**public** **function** addMascota(mascota:Mascota):**void**

**public** **function** delCliente(cliente:Cliente):**void**

## Variables

Este punto se comporta de la misma forma que el estándar en java, donde, excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas (como el caso de los métodos). Los nombres de las variables deben ser cortos pero con significado.

## Prefijos en Variables Globales o que interactúan en clases de interfaz que contengan a más de una clase

En el caso de que se utilicen variables que son únicamente de un objeto, y que son utilizadas en alguna clase de interfaz u otra en que intervienen otras clases u objetos, se utiliza un prefijo, que debe ser el nombre de la clase u objeto, seguido por un ‘\_’ y luego el nombre de la variable.

Ejemplos:

**public** **var** mascota\_nombre:String;

**public** **var** cliente\_nombre:String;

**public** **var** mascota\_edad:String;

**public** **var** cliente\_edad:String;

**ESTANDAR CODIGO DE FLEX**

Estructura de código en flex es de la siguiente forma:

1. Script
   1. Imports
   2. Variables Globales
   3. Funciones
2. Tags de Flex

Ejemplo:

<mx:Panel …">

<mx:Script>

<![CDATA[

**import** mx.controls.Alert;

[**Bindable**]

**public** **var** cliente:Person;

**private** **function** buscarCliente(event:Event):**void**{

getPerson(rutClienteInput.text);

}

]]>

</mx:Script>

<mx:Label x="28" y="28" text="Rut Cliente:" width="84"/>

</mx:Panel>

**Identificadores**

Los componentes son individualizados por sus identificadores y al ser estos importantes para referenciar o ser referenciados por sus funciones, se insertaran al inicio del tag, de la siguiente forma:

<mx:TextInput id="rutClienteInput" x="120" y="26" width="80"/>

**Convenciones de Nombres**

Los identificadores son ocupados principalmente en las entradas y salidas de textos que son utilizados para asignar los atributos del objeto o clase, por ende el identificador en las entradas el nombre del atributo seguido con Input, ejemplo:

Id=”rutInput”

Id=”nombreInput”

En los datagrid no se utiliza identificador para cada columna, sin embargo para poder desplegar un objeto en una fila de la tabla y hacer referencia a que cada atributo aparezca en una columna determinada, se debe cumplir lo siguiente:

La propiedad dataField de la columna de la tabla debe tener el mismo nombre del atributo del objeto que se debe desplegar por ejemplo:

Clase:

**public** **class** Cliente{

**public** **var** nombre:String;

**public** **var** edad:int;

**.**

**.**

}

Datagrid:

<mx:DataGrid …>

<mx:columns>

<mx:DataGridColumn dataField="nombre" …/>

<mx:DataGridColumn dataField="edad" …/>

.

.

</mx:columns>

</mx:DataGrid>