Convenciones de código para el lenguaje de programación JAVA™[1]

1. Introducción

1.1. Por qué convenciones de código

- Las convenciones de código son importantes para los programadores por muchas razones:

- El 80% del coste del código de un programa va a su mantenimiento.

- Casi ningún software lo mantiene toda su vida el auto original.

- Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo.

- Si distribuye su código fuente como un producto, necesita asegurarse de que está bien hecho y presentado como cualquier otro producto.

2. Nombres de archivo

Esta sección enumera las extensiones y los nombres de archivo más usados

2.1. Extensiones de los archivos

El software Java usa las siguientes extensiones para sus archivos

Tipo de archivo	Extensión
Código java	.java
Bytecode de java	.class

Tabla 1

3. Organización de los archivos

Un archivo está formado por secciones que deben estar separadas por líneas en blanco y comentarios opcionales que identifican cada sección. Se deberá evitar la creación de archivos de más de 2000 líneas puesto que son incómodos de manejar.

3.1. Archivos de código fuente Java

[1] Alberto Molpeceres, Convenciones de código para lenguaje de programación JAVA, http://www.javahispano.com, Internet, Acceso: 03/05/2014

Cada archivo fuente Java contiene una única clase o interfaz pública. Cuando algunas clases o interfaces privadas están asociadas a una clase pública, pueden ponerse en el mismo archivo que la clase pública. La clase o interfaz pública debe ser la primera clase o interfaz del archivo.

Los archivos fuente Java tienen la siguiente ordenación:

- Comentarios de inicio
- Sentencias «package» e «import»
- Declaraciones de clases e interfaces

3.1.1. **Comentarios de inicio**

Todos los archivos fuente deben comenzar con un comentario en el que se indican el nombre de la clase, información de la versión, fecha, y copyright

3.1.2. **Sentencias package e import**

La primera línea que no es un comentario de los archivos fuente Java es la sentencia package.

Después de esta, pueden seguir varias sentencias import. Por ejemplo:

- package java.awt;
- import java.awt.peer.CanvasPeer;

3.1.3. **Declaraciones de clases e interfaces**

La siguiente tabla describe las partes de la declaración de una clase o interfaz, en el orden en el que deberían aparecer.

	Partes de la declaración de una clase o interfaz	Notas
1	Comentario de documentación de la clase o interfaz (`/**...*/`)	
2	Sentencia `class` o `interface`	
3	Comentario de implementación de la clase o interfaz si fuera necesario (`/*...*/`)	Este comentario debe contener cualquier información aplicable a toda la clase o interfaz que no sea apropiada para estar en los comentarios de documentación de la clase o interfaz.
4	Variables de clase (`static`)	Primero las variables de clase `public`, después las `protected`, después las de nivel de paquete (sin modificador de acceso) , y después las `private`.
5	Variables de instancia	Primero las `public`, después las `protected`, después las de nivel de paquete (sin modificador de acceso), y después las `private`.
6	Constructores	
7	Métodos	Estos métodos se deben agrupar por funcionalidad más que por ámbito o accesibilidad. Por ejemplo, un método de clase privado puede estar entre dos métodos públicos de instancia. El objetivo es hacer el código mas legible y comprensible.

Ilustración 2

4. Indentación

Se deben emplear cuatro espacios como unidad de indentación. La construcción exacta de la indentación (espacios en blanco o tabuladores) no se especifica. Los tabuladores deben ser exactamente cada 8 espacios (no 4).

4.1. Longitud de la línea

Evite las líneas de más de 80 caracteres ya que no son interpretadas correctamente por muchas terminales y herramientas.

4.2. Rotura de líneas

Cuando una expresión no entre en una línea debe separarla de acuerdo con los siguientes principios:

- Romper después de una coma.
- Romper antes de un operador.
- Preferir roturas 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 línea anterior.

- Si las reglas anteriores provocan que su código parezca confuso o que éste se acumule en el margen derecho

Ejemplo de cómo romper la llamada a un método:

```
unMetodo(expresionLarga1, expresionLarga2, expresionLarga3,
         expresionLarga4, expresionLarga5);

var = unMetodo1(expresionLarga1,
                unMetodo2(expresionLarga2,
                          expresionLarga3));
```

Ilustración 2

5. Comentarios

Los programas Java pueden tener dos tipos de comentarios: comentarios de implementación y comentarios de documentación. Los comentarios de implementación son aquellos que también se encuentran en C++, delimitados por /*...*/, y //.

Los comentarios de documentación (conocidos como «doc comments») existen sólo en Java, y se limitan por /**...*/. Los comentarios de documentación se pueden exportar a archivos HTML con la herramienta javadoc.

Los comentarios de implementación son para comentar nuestro código o para comentarios acerca de una implementación en concreto. Los comentarios de documentación son para describir la especificación del código, libre de una perspectiva de implementación, y para ser leídos por desarrolladores que pueden no tener el código fuente a mano.

Se deben usar los comentarios para dar descripciones de código y facilitar información adicional que no es legible en el código mismo. Los comentarios deben contener sólo información que es relevante para la lectura y entendimiento del programa. Por ejemplo, no se debe incluir como comentario información sobre cómo se construye el paquete correspondiente o en qué directorio reside.

Se podrán tratar discusiones sobre decisiones de diseño que no sean triviales u obvias, pero se debe evitar duplicar información que está presente (de forma clara) en el código ya que es fácil que los comentarios redundantes se queden obsoletos. En general, evite cualquier comentario que pueda quedar obsoleto a medida que el código evoluciona.

Nota: La frecuencia de comentarios a veces refleja la escasez de calidad del código. Cuando se sienta obligado a escribir un comentario considere reescribir el código para hacerlo más claro.

Los comentarios no deben encerrarse en grandes cuadrados dibujados con asteriscos u otros caracteres. Los comentarios nunca deben incluir caracteres especiales como «*backspace*».

5.1. Formatos de los comentarios de implementación

Los programas pueden tener cuatro estilos de comentarios de implementación: de bloque, de una línea, finales, y de fin de línea.

5.1.1. **Comentarios de bloque**

Los comentarios de bloque se usan para dar descripciones de archivos, métodos, estructuras de datos y algoritmos. Los comentarios de bloque se podrán usar al comienzo de cada archivo o antes de cada método. También se pueden usar en otro lugares, tales como el interior de los métodos. Los comentarios de bloque en el interior de una función o método deben ser indentados al mismo nivel que el código que describen

Un comentario de bloque debe ir precedido por una línea en blanco que lo separe del resto del código.

```
/*
 * Aquí hay un comentario de bloque.
 */
```

Ilustración 3

Los comentarios de bloque pueden comenzar con /*-, que es reconocido por indent(1) como el comienzo de un comentario de bloque que no debe ser reformateado. Ejemplo:

```
/*-
 * Aquí tenemos un comentario de bloque con cierto
 * formato especial que quiero que ignore indent(1).
 *
 *    uno
 *        dos
 *            tres
 */
```

Ilustración 4

Nota: Si no se va a utilizar indent(1), no se deberá usar /*- en el código o hacer ninguna otra concesión a la posibilidad de que alguien ejecute indent(1) sobre él. Véase también «Comentarios de documentación

5.1.2. **Comentarios de una línea**

Pueden aparecer comentarios cortos de una única línea al nivel del código que sigue. Si un comentario no se puede escribir en una línea, debe seguir el formato de los comentarios de bloque (sección 5.1.1). Un comentario de una sola línea debe ir precedido de una línea en blanco. Veamos un ejemplo de comentario de una sola línea en código Java:

```
if (condicion) {
    /* Código de la condición. */
    ...
}
```

Ilustración 5

5.1.3. **Comentarios finales**

Pueden aparecer comentarios muy pequeños en la misma línea del código que comentan, pero lo suficientemente alejados de él. Si aparece más de un comentario corto en el mismo trozo de código, deben ser indentados con la misma profundidad. Por ejemplo:

```
if (a == 2) {
    return TRUE;              /* caso especial */
} else {
    return isPrime(a);        /* caso general */
}
```

Ilustración 6

5.1.4. **Comentarios de fin de línea**

El delimitador de comentario // puede convertir en comentario una línea completa o una parte de una línea. No debe ser usado para hacer comentarios de varias líneas consecutivas; sin embargo, puede usarse en líneas consecutivas para comentar secciones de código. Veamos un ejemplo de los tres estilos:

```
if (foo > 1) {
    // Hacer algo.
    ...
} else {
    return false;  // Explicar aquí por que.
}
//if (bar > 1) {
//
//
//    Hacer algo.
//    ...
//}
//else {
//    return false;
//}
```

Ilustración 7

6. Declaraciones

6.1. Cantidad por línea

Se recomienda una declaración por línea, ya que facilita los comentarios. En otras palabras, se prefiere:

```
int nivel; // nivel de indentación
int tam;   // tamaño de la tabla
```

Ilustración 8

6.2. Inicialización

Intente 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 cálculos posteriores.

6.3. Colocación

Sitúe las declaraciones solo al principio de los bloques (un bloque es cualquier código encerrado por llaves «{» y «}»). No espere al primer uso para declararlas; puede confundir al programador incauto y limitar la portabilidad del código dentro de su ámbito.

```
void myMethod() {
    int int1 = 0;       // comienzo del bloque del método

    if (condicion) {
        int int2 = 0;   // comienzo del bloque del "if"
        ...
    }
}
```

Ilustración 9

La excepción a la regla son los índices de bucles for, que en Java se pueden declarar en la sentencia for:

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

Ilustración 10

Evite las declaraciones locales que oculten declaraciones de niveles superiores, por ejemplo, no declare el mismo nombre de variable en un bloque interno:

```
int cuenta;
...
miMetodo() {
    if (condicion) {
        int cuenta = 0;    // EVITAR!
        ...
    }
    ...
}
```

Ilustración 11

6.4. Declaraciones de clases e interfaces

Al programar 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 de declaración
- La llave de cierre «}» empieza una nueva línea indentada ajustada 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() {}
    ...
}
```

Ilustración 12

- Los métodos se separan con una línea en blanco

7. Sentencias

7.1. Sentencias simples

Cada línea debe contener como máximo una sentencia. Ejemplo:

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

Ilustración 13

7.2. Sentencias compuestas

Las sentencias compuestas son sentencias que contienen listas de sentencias encerradas entre llaves «{ sentencias }».

- 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 compuesta; la llave de cierre debe empezar una nueva línea y ser indentada al mismo nivel que el principio de la sentencia compuesta.

- 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 sentencias sin incluir accidentalmente *bugs* por olvidar las llaves.

7.3. Sentencias de retorno

Una sentencia return con un valor no debe usar paréntesis a no ser que que hagan el valor de retorno más obvio de alguna manera. Ejemplo:

7.4. 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;
}
```

Ilustración 14

7.5. Sentencias for

Una sentencia for debe tener la siguiente forma:

```
for (inicializacion; condicion; actualizacion) {
    sentencias;
}
```

Ilustración 15

7.6. Sentencias while

Una sentencia while debe tener la siguiente forma

```
while (condicion) {
    sentencias;
}
```

Ilustración 16

7.7. Sentencias do-while

Una sentencia do-while debe tener la siguiente forma:

```
do {
    sentencias;
} while (condicion);
```

Ilustración 17

7.8. 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;
}
```

Ilustración 18

7.9. Sentencias try-catch

Una sentencia try-catch debe tener la siguiente forma:

```
try {
    sentencias;
} catch (ExceptionClass e) {
    sentencias;
}
```

Ilustración 19

8. Espacio en blanco

8.1. Líneas en blanco

Las líneas en blanco mejoran la facilidad de lectura separando secciones de código que están lógicamente relacionadas. Se deben usar siempre dos líneas en blanco en las siguientes circunstancias:

- Entre las secciones de un archivo fuente
- Entre las definiciones de clases e interfaces.

Se debe usar siempre una línea en blanco en las siguientes circunstancias:

- Entre métodos
- Entre las variables locales de un método y su primera sentencia
- Antes de un comentario de bloque o de un comentario de una línea
- Entre las distintas secciones lógicas de un método para facilitar la lectura

9. Convenciones de nomenclatura

Las convenciones de nomenclatura hacen que el código sea más inteligible al hacerlo más fácil de leer. También pueden dar información sobre la función de un identificador, por ejemplo, cuando es una constante, un paquete o una clase, lo cual puede ser útil para entender el código.

Tipo de identificador	Reglas de nomenclatura	Ejemplos

Paquetes	El prefijo del nombre de un paquete se escribe siempre con letras ASCII en minúsculas, y debe ser uno de los nombres de dominio de alto nivel (actualmente com, edu, gov, mil, net, org) o uno de los códigos ingleses de dos letras que identifican cada país como se especifica en el estándar ISO 3166, 1981. Los siguientes componentes del nombre del paquete variarán de acuerdo a las convenciones de nomenclatura internas de cada organización. Dichas convenciones pueden especificar que algunos nombres de los directorios correspondan a divisiones, departamentos, proyectos o máquinas.	`com.sun.eng`
		`com.apple.quicktime.v2`
		`edu.cmu.cs.bovik.cheese`

Clases	Los nombres de las clases deben ser sustantivos. Cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intente mantener los nombres de las clases simples y descriptivos. Use palabras completas, evite acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como `URL` o `HTML`).	`class Cliente;` `class ImagenAnimada;`
Interfaces	Los nombres de las interfaces siguen la misma regla que las clases.	`interface ObjetoPersistente;` `interface Almacen;`

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.	`ejecutar();`
		`ejecutarRapido();`
		`cogerFondo();`

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. Los nombres de variables no deben empezar con los caracteres *guión bajo* «_» o signo de dólar «$», aunque ambos están permitidos por el lenguaje. Los nombres de las variables deben ser cortos pero significativos. La elección del nombre de una variable debe ser un mnemónico, designado para indicar a un observador ocasional su función. 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; `c`, `d`, y `e` para caracteres.	`int i;`
		`char c;`
		`float miAnchura;`

Constantes	Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un guión bajo («_»). (Las constantes ANSI se deben evitar, para facilitar la depuración.)	`static final int ANCHURA_MINIMA = 4;`
		`static final int ANCHURA_MAXIMA = 999;`
		`static final int COGER_LA_CPU = 1;`

Ilustración 20

10. Hábitos de programación

10.1. Proporcionar acceso a variables de instancia y de clase

No haga pública ninguna variable de instancia o clase sin una buena razón. A menudo las variables de instancia no necesitan ser asignadas (*set*) o consultadas *(get)* explícitamente, esto suele suceder como efecto de una llamada a método.

Un ejemplo de una variable de instancia pública apropiada es el caso en que la clase es esencialmente una estructura de datos, sin comportamiento. En otras palabras, si usara la palabra struct en lugar de una clase (si Java soportara struct), entonces sería adecuado hacer las variables de instancia públicas.

10.2. Referencias a variables y métodos de clase

Evite usar un objeto para acceder a una variable o método de clase (static). Use el nombre de la clase en su lugar. Por ejemplo:

```
metodoDeClase();           //OK
UnaClase.metodoDeClase();  //OK

unObjeto.metodoDeClase();  //EVITAR!
```

Ilustración 21

10.3. Constantes

Las constantes numéricas (literales) no se deben codificar directamente, excepto -1, 0, y 1, que pueden aparecer en un bucle for como contadores.

10.4. Asignaciones de variables

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

```
fooBar.fChar = barFoo.lchar = 'c'; // EVITAR!
```

Ilustración 22

No use el operador de asignación en un lugar donde se pueda confundir con el de igualdad. Ejemplo:

```
if (c++ = d++) { // EVITAR! (Java lo rechaza)
    ...
}
```

Ilustración 23

Se debe escribir:

```
if ((c++ = d++) != 0) {
    ...
}
```

Ilustración 24

No use asignaciones embebidas como un intento de mejorar el rendimiento en tiempo de ejecución. Ese es el trabajo del compilador. Ejemplo:

```
d = (a = b + c) + r; // EVITAR!
```

Ilustración 25

Se debe escribir

```
a = b + c;
d = a + r;
```

Ilustración 25

10.5. Hábitos varios

10.5.1. Paréntesis

En general es una buena idea usar paréntesis en expresiones que involucran a 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.

```
if (a == b && c == d) // EVITAR!

if ((a == b) && (c == d)) // CORRECTO
```

Ilustración 26

10.5.2. Valores de retorno

Intente hacer que la estructura del programa se ajuste a su objetivo. Ejemplo:

```
if (expresionBooleana) {
    return true;
} else {
    return false;
}
```

Ilustración 27

En su lugar se debe escribir

```
return expressionBooleana;
```

Ilustración 28

Del mismo modo,

```
if (condicion) {
    return x;
}
return y;
```

Ilustración 29

Se debe escribir:

```
(x >= 0) ? x : -x;
```

Ilustración 30