

Facultad de Informática

Universidad Politécnica de Madrid

Estándar de Codificación

Ingeniería del Software

Grupo 5

Tabla de contenido

[1. Introducción 3](#_Toc355656076)

[2. Propósito 3](#_Toc355656077)

[3. Notación 3](#_Toc355656078)

[4. Archivos 4](#_Toc355656079)

[4.1. Nombres 4](#_Toc355656080)

[4.2. Organización 4](#_Toc355656081)

[5. Indentación 4](#_Toc355656082)

[5.1. Longitud de líneas 5](#_Toc355656083)

[5.2. Saltos de línea 5](#_Toc355656084)

[6. Comentarios 6](#_Toc355656085)

[6.1. De implementación 6](#_Toc355656086)

[6.2. De documentación 6](#_Toc355656087)

[7. Declaraciones 6](#_Toc355656088)

[7.1. Cantidad 7](#_Toc355656089)

[7.2. Inicialización 7](#_Toc355656090)

[7.3. Ubicación 7](#_Toc355656091)

[7.4. Clases e interfaces 7](#_Toc355656092)

[8. Sentencias 7](#_Toc355656093)

[8.1. Simples 7](#_Toc355656094)

[8.2. Compuestas 8](#_Toc355656095)

[8.3. De retorno 8](#_Toc355656096)

[8.4. Condicionales 8](#_Toc355656097)

[8.5. Repetitivas 9](#_Toc355656098)

[8.6. Try-catch 9](#_Toc355656099)

[9. Espacios en blanco 9](#_Toc355656100)

[9.1. Líneas en blanco 9](#_Toc355656101)

[9.2. Espacios en blanco 10](#_Toc355656102)

[10. Convenciones de nombres 10](#_Toc355656103)

[10.1. Clases e interfaces 10](#_Toc355656104)

[10.2. Métodos 10](#_Toc355656105)

[10.3. Variables 10](#_Toc355656106)

[10.4. Constantes 11](#_Toc355656107)

[11. Referencias 11](#_Toc355656108)

# Introducción

Para permitir la comprensión del código empleado en el desarrollo de manera rápida, fácil y sencilla. Además de garantizar el mantenimiento óptimo de dicho código por parte de los programadores. Se considera redactar este documento con la finalidad de mejorar la productividad y la calidad.

# Propósito

El propósito de este estándar es reunir en un solo documento para los programadores, todas las convenciones, patrones, modelos y formatos que se utilizarán para la creación de código en el lenguaje especificado.

Con la intención de establecer una herramienta de programación común y no inducir a problemas de coherencia en la apertura o modificación de los ficheros fuente, se decide utilizar NetBeans como entorno de desarrollo integrado (IDE) dedicado en exclusiva a lenguaje Java, específicamente se desarrollará bajo entorno JAVA<TM> SE Runtime Environment 1.6 (conocido popularmente como Java6).

# Notación

La notación a utilizar será tipo Camel:

Para los identificadores de clases UpperCamelCase (consiste en escribir los identificadores con la primera letra de cada palabra en mayúscula y el resto en minúscula) y para los métodos y variables LowerCamelCase (primera letra en minúscula). Ej: Coche, FabricaCoches, getMatricula(), numRuedas.

Para las constantes esta notación no es aplicable, se anotarán con todas las letras en mayúscula. Ej: PI, RADIO\_CIRCULO.

Usar nombres cortos en identificadores locales y nombres largos y descriptivos en identificadores globales.

# Archivos

## Nombres

Para los nombres se establecen las siguientes convenciones:

- Los archivos tienen extensión .java

- Las clases tienen extensión .class

- Los nombres de archivos deben ser claros, simples y descriptivos.

Ej: “VideoTienda.java” “Cliente.class”

## Organización

En la organización que debe haber dentro de un archivo de código hay que tener en cuenta tres aspectos, estos son:

- Comentarios de inicio: suministran información sobre el nombre de la clase, la versión y la fecha.

- Sentencia *package* e *import*: son las primeras no comentarios en una clase. La primera de ellas es *package* y después de esta pueden seguir tantas sentencias *import* como sean necesarias.

- Declaración de clases e interfaces: se establece a continuación las partes y el orden en el que estás deben ser descritas:

- Comentario de documentación

- Sentencia *class* o *interface*

- Comentario de implementación

- Variables de clase

- Variables de instancia

- Constructores

- Métodos

# Indentación

Significa mover un bloque de textos a la derecha utilizando espacios en blanco, con el fin de mejorar la legibilidad del código fuente.

Se recomienda usar 4 espacios en blanco como unidad de indentación.

## Longitud de líneas

Con el fin de evitar errores en la comprensión por parte de algunas consolas o programas se sugiere que las líneas de código no excedan en el número de caracteres recomendados por línea (80 caracteres).

## Saltos de línea

Cuando para una expresión no es suficiente una sola línea, debido a su longitud, se continúa en la línea siguiente, teniendo en cuenta los siguientes criterios que se muestran a continuación:

- Después de una coma:

enviarCliente(getCodigo(), getNombre(), getApellido(),

getDireccion(), getTelefono());

- Antes de un operador:

total = cantidad\*(valor + recargo

–descuento) + 100;

- Saltos de alto nivel en lugar de saltos de bajo nivel

alquiler = datosAlquiler(getCodigoCliente(),

datosPelicula(getCodigoP(),

getEjemplar()));

- Si todo lo anterior confunde el código, indentar a ocho espacios

If((condicion1 && condicion2)

|| (condicion3 && condicion4)

|| (condicion5 && condicion6)){

Sentencias;

}

# Comentarios

Existen dos tipos de comentarios: de implementación y de documentación. Los primeros se usan para comentar el código a través de los caracteres /\*…\*/ y //. Sin embargo los de documentación se usan para describir especificaciones del código usando los caracteres /\*\*…\*/. Recomendaciones:

- Claros

- Apropiados

- Evitar la redundancia

- No incluir caracteres especiales

- No hacer uso excesivo de asteriscos u otros caracteres en su diseño

## De implementación

Se clasifican en:

- Comentarios de bloque: Son usados al comienzo de cada archivo, antes de cada método o en el interior de estos, con el fin de dar una descripción de los mismos. Están precedidos por una línea en blanco.

- Comentarios de una línea: son comentarios cortos indentados al mismo nivel de la línea de código que sigue. Están precedidos por una línea en blanco.

- Comentarios de aclaración: aparecen en la misma línea del código comentado. Se escriben a una distancia prudente de tal manera que no confundan al código mismo. Cuando se escribe más de un comentario aclaratorio estos deben ser indentados al mismo nivel.

## De documentación

Describen las clases, interfaces, constructores, métodos y atributos. Van siempre ubicados antes de la declaración y con los caracteres /\*\*…\*/

Este tipo de comentarios no se escriben dentro del bloque de un método o constructor, ya que Java asocia el comentario con la primera sentencia que se encuentra después de éste.

# Declaraciones

## Cantidad

Se recomienda hacer una sola declaración por línea, ya que favorece el uso de comentarios. Por otro lado se sugiere no hacer declaraciones de diferentes tipos en la misma línea.

## Inicialización

Al declarar las variables, se hace necesario inicializarlas inmediatamente en el lugar en el que han sido declaradas. Si esta acción no se efectúa, debe ser sólo porque el valor inicial de dicha variable depende de un cálculo u operación a realizar.

## Ubicación

Las declaraciones se ubicarán siempre al principio de cada bloque excepto cuando se hace uso de índices en bucles como el *for*.

Evitar declaraciones locales que oculten declaraciones de un nivel más alto.

## Clases e interfaces

Se deben tener en cuenta los siguientes aspectos:

- Debe existir al menos una línea en blanco que separe los métodos.

- No dejar espacios en blanco entre el nombre del método y el paréntesis de la lista de parámetros.

- La llave de apertura “{” debe estar ubicada al final de línea de declaración de sentencia.

- La de cierre “}” se indenta al comienzo de la línea, de tal manera que se alinee con la sentencia de apertura, excepto cuando se trata de un método vacío.

# Sentencias

## Simples

Se recomienda que cada línea contenga una sola sentencia.

## Compuestas

Son los conjuntos de sentencias que se encuentran entre llaves “{}”. Tener en cuenta las reglas de indentación según el nivel de la sentencia, la ubicación de las llaves. Evitar el olvido de las llaves de cierre.

## De retorno

En el uso de la sentencia *return* no debe usarse paréntesis “()”, a menos que el valor devuelto sea el resultado de alguna operación.

return s.substring(0,i); return(alquiler+iva);

## Condicionales

- If, if-else: Esta sentencia siempre lleva llaves “{}” en cualquiera de sus casos:

**Sentencia if Sentencia if-else Sentencia if else-if else**

if (condición) { if (condición) { if (condición) {

sentencias; sentencias; sentencias;

} }else { }else if (condición){

sentencias; sentencias;

} }else {

sentencias;

}

- Switch: Incluye un caso por defecto. El *break* interrumpe el flujo de la sentencia evitando propagación por error ante nuevos casos. Su estructura es:

switch (condicion){

case ABC:

sentencias;

break;

case DEF:

sentencias;

break;

default:

sentencias;

break;

)

## Repetitivas

- For: la sentencia *for* debe tener la siguiente estructura:

for(inicialización; condición; actualización){

Sentencias;

}

- While: Su estructura es la siguiente:

while(condicion){

sentencias;

}

- Do-while: Debe tener la siguiente estructura:

do{

sentencias;

}while(condicion);

## Try-catch

La sentencia para la captura de excepciones debe estructurarse de este modo:

try{

sentencias;

}cath(ExceptionClass e){

sentencias;

}

# Espacios en blanco

Pueden darse de dos formas:

## Líneas en blanco

Separan secciones de código de la siguiente manera:

• Una línea:

- Entre dos métodos.

- Entre variables locales y la primera sentencia de un método.

- Antes de un comentario de bloque o de una línea.

• Dos líneas:

- Entre secciones de un archivo fuente.

- Entre las definiciones de clases e interfaces.

## Espacios en blanco

Separan elementos dentro de las líneas de código, de este modo:

- Palabra clave seguida de paréntesis, no aplica para los métodos.

- Después de cada coma en una lista de argumentos

- Antes y después de cada operador excepto el “.” y los de incremento o decremento.

- En las expresiones dentro de un *for*.

# Convenciones de nombres

El nombre suministra información acerca de la función de un identificador por lo que ayuda a comprender el programa.

## Clases e interfaces

A la hora de nombrar a una clase o a una interfaz se deben tener en cuenta los siguientes aspectos:

- El nombre debe ser un sustantivo.

- Nombre simple y descriptivo.

- Evitar abreviaturas.

## Métodos

A la hora de nombrar a un método se deben tener en cuenta los siguientes aspectos:

- El nombre debe ser un verbo.

- Nombre simple y descriptivo.

- Evitar abreviaturas.

## Variables

A la hora de nombrar a una variable se deben tener en cuenta los siguientes aspectos:

- Se escriben en minúscula.

- No deben iniciar con “\_”, ni “$”

- Nombre corto y con significado.

- Nombre mnemotécnico.

- Evitar nombres de un solo carácter excepto para variables índices {i, j}.

## Constantes

A la hora de nombrar a una constante se deben tener en cuenta los siguientes aspectos:

- Se escriben en mayúscula.

- Si es compuesta, las palabras que la componen se separan con”\_”.

- Nombre corto y con significado.

- Nombre mnemotécnico.

# Referencias

- Documento basado en las “Convenciones de Código para el lenguaje de programación Java” revisado 10 Mayo del 2001 por Scott Hommel (Sun Microsystems Inc.)

- Code Conventions for the Java Programming Language (Oracle)