UNIVERSIDAD AUTÓNOMA DE YUCATÁN

Facultad de Matemáticas
Licenciatura en Ingeniería de Software

Asignatura:
Construcción de Software [Acompañamiento]

Documento de Estándar de Programación

Equipo:
Eleani Consuelo Moo Sosa
Herve Ordaz Osorio
Diego Flores Mis

Docente:
Edwin León Bojórquez

Fecha de Entrega:
22 /Febrero/2024

Índice

Estándar de Programación: Convenciones de código para el lenguaje de programación Java

1. Ficheros

1.1 Extensiones de los ficheros

El software Java usa las siguientes extensiones para los ficheros:

Tipo de fichero	Extensión
Fuente Java	.java
Bytecode de java	.class

1.2 Nombre de ficheros comunes

Los nombres de ficheros más utilizados incluyen:

Nombre de fichero	Uso
GNUmakefile	El nombre preferido para dicheros “make”. Usamos *gnumake* para construir nuestro software.
README	El nombre preferido para el fichero que resume los contenidos de un directorio particular.

2. Organización de los ficheros

2.1 Comentarios de comienzo

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

```
/*
 * Nombre de la clase
 *
 * Informacion de la version
 *
 * Fecha
 *
 * Copyright
 */
```

2.1.1 Sentencias package e import

La primera línea no-comentario de los ficheros fuente Java es la sentencia *package.* Después de esta, pueden seguir varias sentencias *import.* Por ejemplo:

```
package java.awt;

import java.awt.peer.CanvasPeer;
```

Nota: El primer componente del nombre de un paquete único se escribe siempre en minúsculas con caracteres ASCII y debe ser uno de los nombres de dominio de último nivel, actualmente com, edu, gov, mil, net, org, o uno de los códigos ingleses de dos letras que especifican el pais como se define en el ISO Standard 3166, 1981.

2.1.2 Declaración de clases e interfaces

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

	Partes de la declaración de una clase o interface	Notas
1	Comentario de documentación de la clase o interface (/**…*/)	Ver
2	Sentencia *class* o *interface.*	
3	Comentario de implementación de la clase o interface si fuera necesario (/*…*/)	Este comentario debe contener cualquier información aplicable a toda la clase o interface que no era apropiada para estar en los comentarios de documentación de la clase o interface.
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 visión 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.

3. Indentación

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

3.1 Longitud de la línea

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

Nota: Ejemplos para uso en la documentación deben tener una longitud inferior, generalmente no más de 70 caracteres.

3.2 Rompiendo líneas

Cuando una expresión no entre en una línea, romperla de acuerdo con estos 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 llevan a código confuso o a código que se aglutina en el margen derecho, indentar justo 8 espacios en su lugar.

4. 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 ficheros HTML con la herramienta javadoc.

Los comentarios de implementación son para comentar nuestro código o para comentarios acerca de una implementación particular. 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, información sobre cómo se construye el paquete correspondiente o en que directorio reside no debe ser incluida como comentario.

Son apropiadas las discusiones sobre decisiones de diseño no triviales o no obvias, pero evitar duplicar información que está presente (de forma clara) en el código ya que es fácil que los comentarios redundantes se queden desfasados. En general, evitar cualquier comentario que pueda quedar desfasado a medida que el código evoluciona.

Nota: La frecuencia de comentarios a veces refleja una pobre 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.

4.1 Formatos de los comentarios de implementación

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

4.1.1 Comentarios de bloque

Los comentarios de bloque se usan para dar descripciones de ficheros, métodos, estructuras de datos y algoritmos. Los comentarios de bloque se podrán usar al comienzo de cada fichero o antes de cada método. También se pueden usar en otros 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 cometario de bloque debe ir precedido por una línea en blanco que lo separe del resto del código.

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

Los comentaros 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:

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

Nota: Si no se usa indent(1), no se tiene que usar /*- en el código o hacer cualquier otra concesión a la posibilidad de que alguien ejecute indent(1) sobre él.

4.1.2 Comentarios de una línea

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

```
if (condicion) {

    /* Código de la condicion. */
    ...
}
```

4.1.3 Comentarios de remolque

Pueden aparecer comentarios muy pequeños en la misma línea que describen, pero deben ser movidos lo suficientemente lejos para separarlos de las sentencias. Si más de un comentario corto aparecen en el mismo trozo de código, deben ser indentados con la misma profundidad.

Aquí un ejemplo de comentario de remolque:

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

4.1.4 Comentarios de fin de línea

El delimitador de comentario // puede convertir en comentario una línea completa o una parte de una linea. No debe ser usado para hacer comentario de varias líneas

consecutivas; sin embargo, puede usarse en líneas consecutivas para comentar secciones de código. Aquí tienen ejemplos de los tres estilos:

```
if (foo > 1) {

    // Hacer algo.
    ...
}
else {
    return false;          // Explicar aqui por que.


}
//if (bar > 1) {
//
//    // Hacer algo.
//    ...
//}
//else {
//    return false;
//}
```

4.2 Comentarios de documentación

Los comentarios de documentación describen clases Java, interfaces, constructores, métodos y atributos. Cada comentario de documentación se encierra con los delimitadores de comentarios /**...*/, con un comentario por clase, interface o miembro (método o atributo). Este comentario debe aparecer justo antes de la declaración:

```
/**
 * La clase Ejemplo ofrece ...
 */
public class Ejemplo { ...
```

Darse cuenta de que las clases e interfaces de alto nivel son están indentados, mientras que sus miembros los están. La primera línea de un comentario de documentación (/**) para clases e interfaces no está indentada, subsecuentes líneas tienen cada una un espacio de indentación (para alinear verticalmente los asteriscos). Los miembros, incluidos los constructores, tienen cuatro espacios para la primera línea y 5 para las siguientes.

Si se necesita dar información sobre una clase, interface, variable o método que no es apropiada para la documentación, usar un comentario de implementación de bloque o de una línea para comentarlo inmediatamente después de la declaración. Por ejemplo, detalles de implementación de una clase deben ir en un comentario de implementación de bloque *siguiendo* a la sentencia *class*, no en el comentario de documentación de la clase.

Los comentarios de documentación no deben colocarse en el interior de la definición de un método o constructor, ya que Java asocia los comentarios de documentación con la primera declaración después del comentario.

5. Declaraciones

5.1 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
```

Antes que:

```
int level, size;
```

No poner diferentes tipos en la misma línea. Ejemplo:

```
int foo,  fooarray[]; //ERROR!
```

Nota: Los ejemplos anteriores usan un espacio entre el tipo y el identificador. Una alternativa aceptable es usar tabuladores, por ejemplo:

```
int      level;          // nivel de indentacion
int      size;            // tamaño de la tabla
Object   currentEntry;   // entrada de la tabla seleccionada
actualmente
```

5.2 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.

5.3 Colocación

Poner 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 a programadores no preavisados 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 índices de bucles *for*, que en Java se pueden declarar en la sentencia *for:*

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

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

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

5.4 Declaraciones de class e interfaces

Al codificar clases o 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 las sentencia de declaración.
- 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 apertura "{".
- Los métodos se separan con una línea en blanco.

6. Sentencias

6.1 Sentencias simples

Cada línea debe contener como mucho una sentencia. Ejemplo:

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

6.2 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 linea 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 snecillo añadir sentencias sin incluir bugs accidentalmente por olvidar las llaves.

6.3 Sentencias de retorno

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

```
return;

return miDiscoDuro.size();

return (tamanyo ? tamanyo : tamanyoPorDefecto);
```

6.4 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;
}
```

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

```
if (condicion) //EVITAR! ESTO OMITE LAS LLAVES {}!
    sentencia;
```

6.5 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, condición y actualización) debe tener la siguiente forma:

```
for (inicializacion; condicion; actualizacion);
```

Al usar el operador coma en la clausula de inicialización o actualización de una sentencia for, evitar la complejidad de usar más de tres variables. Si se necesita, usar sentencias separadas antes de bucle *for* (para la cláusula de inicialización) o

al final del bucle (para la cláusula de actualización).

6.6 Sentencias while

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

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

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

```
while (condicion);
```

6.7 Sentencias do-while

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

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

6.8 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;
}
```

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 previene que se propague por error si luego se añade otro caso.

6.9 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 halla completado con éxito o no.

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

7. Espacios en blanco

7.1 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 fichero 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.

7.2 Espacios en blanco

Se deben usar espacios en blanco en las siguientes circunstancias:

- Una palabra clave del lenguaje seguida por un paréntesis debe separarse por un espacio.
- Debe aparecer un espacio en blanco después de cada coma en las listas de argumentos.
- Todos los operadores binarios excepto “.” se deben separar de sus operandos con espacios en blanco. Los espacios en blanco no deben separar los operadores unarios, incremento (“++”) y decremento (“—“) de sus operandos.

- Las expresiones en una sentencia *for* se deben separar con espacios en blanco.
- Los casteos deben ir seguidos de un espacio en blanco.

8. Convenciones de nombres

Las convenciones de nombres hacen los programas más entendibles haciéndolos 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, que puede ser útil para entender el código.

Tipos de identificadores	Reglas para nombramientos	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 identifícan cada país como se especifica en el ISO Standard 3166, 1981. Los subsecuentes componentes del nombre del paquete variarán de acuerdo con las convenciones de nombres 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	class Cliente; class ImagenAnimada;

	primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases simples y descriptivos. Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo, como URL o HTML).	
Interfaces	Los nombres de las interfaces siguen la misma regla que las clases.	interface ObjetoPresistente; 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.	obtenerFondo();
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 subguión "_" o signo del dólar "$", aunque ambos están permitidos por el lenguaje.	

	Los nombres de las variables deben ser cortos, pero con significado. La elección del nombre de una variable debe ser un mnemónico, designado para indicar a un observador casual 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.	

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