Convenciones del código Java

1 - Introducción

1.1 ¿Por qué tener convenciones de código?

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

- El 80% del coste de vida útil de un software se destina al mantenimiento.
- Casi ningún software recibe mantenimiento durante toda su vida por parte del autor original.
- Las convenciones de código mejoran la legibilidad del software, lo que permite a los ingenieros comprender el código nuevo más rápida y completamente.
- Si envía su código fuente como producto, debe asegurarse de que esté tan bien empaquetado y limpio como cualquier otro producto que cree.

Para que las convenciones funcionen, todos los desarrolladores de software deben cumplirlas. Todos.

1.2 Agradecimientos

Este documento refleja los estándares de codificación del lenguaje Java actuales. *Yo*d*A V*en *a*el *Yo*h *a*mi *Especificación del idioma*F, de Sun Microsystems, Inc. Las principales contribuciones provienen de Peter King, Patrick Naughton, Mike DeMoney, Jonni Kanerva, Kathy Walrath y Scott Hommel.

Este documento es mantenido por Scott Hommel. Los comentarios deben enviarse a shommel@eng.sun.com

2 - Nombres de archivos

En esta sección se enumeran los sufijos y nombres de archivos más utilizados.

2.1 Sufijos de archivos

El software Java utiliza los siguientes sufijos de archivo:

Tipo de archivo	Sufijo
Código fuente de Java	.java
Código de bytes de Java	. clase

2.2 Nombres de archivos comunes

Los nombres de archivos utilizados con frecuencia incluyen:

Nombre del archivo	Usar
Archivo GNUmake	El nombre preferido para los archivos make. Usamosgnumakepara construir nuestro software.
LÉAME	El nombre preferido para el archivo que resume el contenido de un directorio particular.

3 - Organización de archivos

Un archivo consta de secciones que deben estar separadas por líneas en blanco y un comentario opcional que identifique cada sección.

Los archivos con más de 2000 líneas son engorrosos y deben evitarse.

Para ver un ejemplo de un programa Java correctamente formateado, consulte “Ejemplo de archivo fuente Java” en la página 18.

3.1 Archivos fuente de Java

Cada archivo fuente de Java contiene una única clase o interfaz pública. Cuando se asocian clases e interfaces privadas a una clase pública, se pueden incluir en el mismo archivo fuente que la clase pública. La clase pública debe ser la primera clase o interfaz del archivo.

Los archivos fuente de Java tienen el siguiente orden:

- Comentarios iniciales (ver “Comentarios iniciales” en la página 2)
- Declaraciones de paquetes e importaciones
- Declaraciones de clases e interfaces (consulte “Declaraciones de clases e interfaces” en la página 3)

3.1.1 Comentarios iniciales

Todos los archivos fuente deben comenzar con un comentario de estilo C que incluya el nombre de la clase, la información de la versión, la fecha y el aviso de derechos de autor:

```
/*
 * Nombre de la clase
 *
 * Información de la versión
 *
 * Fecha
 *
 * Aviso de derechos de autor
 */
```

3.1.2 Declaraciones de paquetes e importaciones

La primera línea que no es un comentario de la mayoría de los archivos fuente de JavaPensilvania esckaedaddeclaración. Después de eso, importarPueden seguirse declaraciones. Por ejemplo:

```
paquete java.awt;

importar java.awt.peer.CanvasPeer;
```

3.1.3 Declaraciones de clases e interfaces

La siguiente tabla describe las partes de una declaración de clase o interfaz, en el orden en que deben aparecer. Consulte “Ejemplo de archivo fuente Java” en la página 18 para ver un ejemplo con comentarios.

	Parte de la declaración de clase/interfaz	**Notas**
1	Comentario de documentación de clase/interfaz /(**...*/)	Consulte “Comentarios de la documentación” en la página 8 para obtener información sobre lo que debe contener este comentario.
2	claseointerfaz declaración	
3	Comentario de implementación de clase/interfaz /(*...*/), si es necesario	Este comentario debe contener cualquier información de toda la clase o de toda la interfaz que no sea apropiada para el comentario de la documentación de la clase/ interfaz.
4	Clases(estático)variables	Primero elpúblicovariables de clase, entonces lapagmi ro- protegido,luego el nivel del paquete (sin modificador de acceso), y luego elprivado .
5	variables de instancia	Abetopúblico,entoncesprotegido,luego el nivel del paquete (sin modificador de acceso), y elpagmionorte ivate .
6	Constructores	
7	métodos	Estos métodos deben agruparse por funcionalidad, no por alcance o accesibilidad. Por ejemplo, un método de clase privado puede estar entre dos métodos de instancia públicos. El objetivo es facilitar la lectura y la comprensión del código.

4 - Sangría

Se deben usar cuatro espacios como unidad de sangría. La construcción exacta de la sangría (espacios vs. tabulaciones) no está especificada. Las tabulaciones deben colocarse exactamente cada 8 espacios (no 4).

4.1 Longitud de línea

Evite líneas de más de 80 caracteres, ya que muchas terminales y herramientas no las gestionan bien.

Nota: Los ejemplos para usar en la documentación deben tener una longitud de línea más corta (generalmente no más de 70 caracteres).

4.2 Líneas de envoltura

Cuando una expresión no quepa en una sola línea, divídala según estos principios generales:

• Salto después de una coma.

• Romper delante de un operador.

• Prefiera los descansos de nivel superior a los de nivel inferior.

• Alinee la nueva línea con el comienzo de la expresión al mismo nivel de la línea anterior.

• Si las reglas anteriores generan un código confuso o que queda apretado contra el margen derecho, simplemente sangre 8 espacios en su lugar.

A continuación se muestran algunos ejemplos de llamadas a métodos que interrumpen el proceso:

```
algúnMétodo(ExpresiónLarga1, ExpresiónLarga2, ExpresiónLarga3,
        longExpression4, longExpression5);

var = algúnMétodo1(expresiónlarga1,
                algúnMétodo2(Expresiónlarga2,
                        longExpression3));
```

A continuación se presentan dos ejemplos de división de una expresión aritmética. Se prefiere el primero, ya que la división se produce fuera de la expresión entre paréntesis, que se encuentra en un nivel superior.

```
nombreLargo1 = nombreLargo2 * (nombreLargo3 + nombreLargo4 - nombreLargo5)
            + 4 * longname6; // PREFER

nombreLargo1 = nombreLargo2 * (nombreLargo3 + nombreLargo4
                        - longName5) + 4 * longName6; // EVITAR
```

A continuación se presentan dos ejemplos de declaraciones de métodos con sangría. El primero es el caso convencional. El segundo, si se usara la sangría convencional, desplazaría la segunda y la tercera línea hacia la derecha, por lo que solo sangraría 8 espacios.

```
//SANGRÍA CONVENCIONAL
algúnMétodo(int unArg, Objeto otroArg, String otroArg,
        Objeto y otro más) {
    ...
}

//SANGRIENTA 8 ESPACIOS PARA EVITAR SANGRIENTA MUY PROFUNDA
private static synchronized horkingLongMethodName(int anArg,
        Objeto otroArg, Cadena otroArg, Objeto
        y otroArg) {
    ...
}
```

Ajuste de línea parasiLas declaraciones generalmente deben usar la regla de los 8 espacios, ya que la sangría convencional (4 espacios) dificulta la lectura del cuerpo. Por ejemplo:

```
//NO UTILICE ESTA SANGRÍA si
((condición1 y condición2)
    || (condición3 y condición4) ||!(condición5 y
    condición6)) { //MALAS ENVOLTURAS
    doSomethingAboutIt();               //HAZ QUE ESTA LÍNEA SEA FÁCIL DE PASAR POR ALTO
}

//UTILICE ESTA SANGRÍA EN SU LUGAR si
((condición1 && condición2)
        || (condición3 && condición4) ||!
        (condición5 && condición6))
        { doSomethingAboutIt();
}

//O USA ESTO
si ((condición1 && condición2) || (condición3 && condición4)
        ||!(condición5 y condición6))
        { hazAlgoSobreEllo();
}
```

A continuación se muestran tres formas aceptables de formatear expresiones ternarias:

```
alfa = (aLongBooleanExpression) ? beta : gamma;

alfa = (aLongBooleanExpression) ? beta
                                : gama;

alfa = (unaExpresiónBooleanaLarga)
        ? beta
        : gama;
```

5 - Comentarios

Los programas Java pueden tener dos tipos de comentarios: de implementación y de documentación. Los comentarios de implementación son los de C++, delimitados por /*...*/ y //. Los comentarios de documentación (conocidos como "comentarios de documentación") son exclusivos de Java y están delimitados por /*y*...*/. Los comentarios de documentación se pueden extraer a archivos HTML con la herramienta javadoc.

Los comentarios de implementación sirven para comentar el código o sobre la implementación específica. Los comentarios de documentación describen la especificación del código, desde una perspectiva independiente de la implementación, para que los desarrolladores, aunque no tengan el código fuente a mano, puedan leerlos.

Los comentarios deben usarse para ofrecer una visión general del código y proporcionar información adicional que no esté disponible en el propio código. Deben contener únicamente información relevante para la lectura y comprensión del programa. Por ejemplo, no debe incluirse como comentario información sobre cómo se compila el paquete correspondiente ni en qué directorio se encuentra.

Es apropiado discutir decisiones de diseño no triviales o no obvias, pero evite duplicar información presente en el código (y que se desprende claramente del mismo). Es muy fácil que los comentarios redundantes queden obsoletos. En general, evite cualquier comentario que pueda quedar obsoleto a medida que el código evoluciona.

Nota: La frecuencia de los comentarios a veces refleja la mala calidad del código. Cuando sienta la necesidad de añadir un comentario, considere reescribir el código para que sea más claro.

Los comentarios no deben encerrarse en grandes cuadros con asteriscos u otros caracteres. Nunca deben incluir caracteres especiales como el salto de página o la tecla de retroceso.

5.1 Formatos de comentarios de implementación

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

5.1.1 Comentarios en bloque

Los comentarios de bloque se utilizan para describir archivos, métodos, estructuras de datos y algoritmos. Pueden usarse al principio de cada archivo y antes de cada método. También pueden usarse en otros lugares, como dentro de los métodos. Los comentarios de bloque dentro de una función o método deben tener la misma sangría que el código que describen.

Un comentario de bloque debe estar precedido por una línea en blanco para diferenciarlo del resto del código.

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

Los comentarios en bloque pueden comenzar con /t*h- , lo cual se reconoce por**i**y**sangría**(1) como inicio de un comentario en bloque que no debe reformatearse. Ejemplo:

```
/*-
 *Aquí hay un comentario en bloque con algunos comentarios muy especiales.
 * formato que quiero que indent(1) ignore.
 *
 * uno
 * dos
 *          tres
 * /
```

Nota: Si no lo usas**sangrar**(1), no tienes que usarnos * - en tu código o hacer cualquier otro concesiones a la posibilidad de que alguien más pudiera**en**el**d**Ruso**mi**norte**el**(1) en su código.

Consulte también “Comentarios sobre la documentación” en la página 8.

5.1.2 Comentarios de una sola línea

Los comentarios cortos pueden aparecer en una sola línea, con sangría al nivel del código que sigue. Si un comentario no puede escribirse en una sola línea, debe seguir el formato de comentario de bloque (véase la sección 5.1.1). Un comentario de una sola línea debe ir precedido de una línea en blanco. A continuación, se muestra un ejemplo de un comentario de una sola línea en código Java:

```
si (condición) {

    /* Manejar la condición.
    */ . . .
}
```

5.1.3 Comentarios finales

Los comentarios muy breves pueden aparecer en la misma línea que el código que describen, pero deben estar lo suficientemente separados de las declaraciones. Si aparece más de un comentario breve en un fragmento de código, todos deben tener la misma sangría de tabulación.

A continuación se muestra un ejemplo de un comentario final en el código Java:

```
si (a == 2) {
    devuelve VERDADERO;             /* caso especial */
} demás {
    devolver esPrime(a);        /* funciona solo para impares */
}
```

5.1.4 Comentarios de fin de línea

El delimitador de comentarios // permite comentar una línea completa o solo una parte. No debe usarse en varias líneas consecutivas para comentarios de texto; sin embargo, sí puede usarse en varias líneas consecutivas para comentar secciones de código. A continuación, se muestran ejemplos de los tres estilos:

```
si (foo > 1) {

    // Haz una doble
    voltereta...
}
demás{
    devuelve falso;             //Explica por qué aquí.
}
```

```
//si (barra > 1) { //

//    // Haz un triple
//    flip...
//}
//demás{
// devuelve falso;
//}
```

5.2 Comentarios sobre la documentación

Nota: Consulte “Ejemplo de archivo fuente Java” en la página 18 para ver ejemplos de los formatos de comentarios descritos aquí.

Para obtener más detalles, consulte “Cómo escribir comentarios de documentos para Javadoc”, que incluye información sobre la etiqueta de comentario de documentación (retorno, @param, @ver):

http://java.sun.com/products/jdk/javadoc/writingdoccomments.html

Para obtener más detalles sobre los comentarios de documentos y javadoc, consulte la página de inicio de javadoc en:

http://java.sun.com/products/jdk/javadoc/

Los comentarios de documentación describen clases, interfaces, constructores, métodos y campos de Java. Cada comentario de documentación se establece dentro del delimitador de comentarios/*e*r..s.*/, con un comentario por clase, interfaz o miembro. Este comentario debe aparecer justo antes de la declaración:

```
/**
 * La clase Ejemplo proporciona...
 * /
clase pública Ejemplo { ...
```

Tenga en cuenta que las clases e interfaces de nivel superior no tienen sangría, mientras que sus miembros sí. La primera línea del comentario de documentación/*(*) para clases e interfaces no tiene sangría; las líneas de comentario de documentación subsiguientes tienen un espacio de sangría cada una (para alinear verticalmente los asteriscos). Los miembros, incluidos los constructores, tienen 4 espacios para la primera línea de comentario de documentación y 5 espacios a partir de entonces.

Si necesita proporcionar información sobre una clase, interfaz, variable o método que no es apropiado para la documentación, utilice un comentario de bloque de implementación (consulte la sección 5.1.1) o un comentario de una sola línea (consulte la sección 5.1.2) inmediatamente.*a*te*pie*yo*mi*y*o*La declaración. Por ejemplo, los detalles sobre la implementación de una clase deben incluirse en un comentario del bloque de implementación. *siguiente*la declaración de la clase, no en el comentario del documento de la clase.

Los comentarios de documentación no deben ubicarse dentro de un bloque de definición de método o constructor, porque Java asocia los comentarios de documentación con la primera declaración.*Fo**el*a *mi*el*o*Otro comentario.

6 - Declaraciones

6.1 Número por línea

Se recomienda una declaración por línea, ya que fomenta los comentarios. En otras palabras,

```
int nivel; // nivel de sangría int
tamaño; // tamaño de la tabla
```

se prefiere sobre

```
int nivel, tamaño;
```

No coloque diferentes tipos en la misma línea. Ejemplo:

```
int foo, fooarray[]; //¡INCORRECTO!
```

Nota: Los ejemplos anteriores utilizan un espacio entre el tipo y el identificador. Otra alternativa aceptable es usar tabulaciones, por ejemplo:

```
entero      nivel;          // nivel de sangría
entero      tamaño;         // tamaño de la tabla
Objeto      currentEntry; // entrada de la tabla seleccionada actualmente
```

6.2 Inicialización

Intenta 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 algún cálculo que se realice primero.

6.3 Colocación

Coloque las declaraciones solo al principio de los bloques. (Un bloque es cualquier código entre llaves "{" y "}"). No espere a declarar las variables hasta su primer uso; esto puede confundir al programador inexperto y dificultar la portabilidad del código dentro del ámbito.

```
void miMétodo() {
    entero int1 = 0;        // comienzo del bloque de método

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

La única excepción a la regla son los índices.paraoobucles f, que en Java se pueden declarar enFelohoDeclaración electrónica:

```
para (int i = 0; i < maxLoops; i++) { ... }
```

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

```
int contar;
...
miMétodo() {
    si (condición) {
        int contar;       // ¡EVITAR!
        ...
    }
    ...
}
```

6.4 Declaraciones de clases e interfaces

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

• No debe haber espacio entre el nombre de un método y el paréntesis "(" que inicia su lista de parámetros

• La llave de apertura "{" aparece al final de la misma línea que la declaración

• La llave de cierre "}" inicia una línea por sí sola, sangrada para que coincida con su declaración de apertura correspondiente, excepto cuando se trata de una declaración nula, el "}" debe aparecer inmediatamente después del "{"

```
clase Sample extiende Object {
    entero ivar1;
    entero ivar2;

    Muestra(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int métodovacío() {}

    ...
}
```

• Los métodos están separados por una línea en blanco.

7 - Declaraciones

7.1 Declaraciones simples

Cada línea debe contener como máximo una declaración. Ejemplo:

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

7.2 Declaraciones compuestas

Las declaraciones compuestas son declaraciones que contienen listas de declaraciones encerradas entre llaves " { declaraciones } ". Consulte las siguientes secciones para ver ejemplos.

- Las declaraciones adjuntas deben tener una sangría un nivel más que la declaración compuesta.
- La llave de apertura debe estar al final de la línea que inicia la declaración compuesta; la llave de cierre debe iniciar una línea y estar sangrada hasta el comienzo de la declaración compuesta.

- Se utilizan llaves alrededor de todas las declaraciones, incluso de declaraciones individuales, cuando son parte de una estructura de control, comosi-además oparaDeclaración. Esto hace que sea más fácil agregar declaraciones sin introducir errores accidentalmente debido a olvidarse de agregar llaves.

7.3 Declaraciones de retorno

AdevolverLas declaraciones con un valor no deben usar paréntesis a menos que hagan más evidente el valor de retorno. Ejemplo:

```
devolver;

devuelve myDisk.size();

devolver (tamaño ? tamaño : tamañoPredeterminado);
```

7.4 Declaraciones if, if-else, if else-if else

Elsi-si no La clase de declaraciones debe tener la siguiente forma:

```
si (  condición  ) {
      declaraciones  ;
}

si (  condición  ) {
      declaraciones  ;
} demás {
      declaraciones  ;
}

si (condición  ) {
      declaraciones  ;
} de lo contrario si (  condición  ) {
      declaraciones  ;
} demás {
      declaraciones  ;
}
```

Nota:si Las sentencias siempre usan llaves {}. Evite la siguiente forma propensa a errores:

```
si (  condición  ) //¡EVITAR! ESTO OMITIÓ LAS LLAVES {}! ;
      declaración
```

7.5 para declaraciones

AparaLa declaración debe tener el siguiente formato:

para (*inicialización* ;*condición* ;*actualizar*) {
declaraciones ;
}

Un vacíoparaLa declaración (aquella en la que se realiza todo el trabajo en las cláusulas de inicialización, condición y actualización) debe tener la siguiente forma:

para (*inicialización* ;*condición* ;*actualizar*);

Al utilizar el operador de coma en la cláusula de inicialización o actualización parade una declaración, evitar La complejidad de usar más de tres variables. Si es necesario, utilice declaraciones separadas antes de lapara bucle (para la cláusula de inicialización) o al final del bucle (para la cláusula de actualización).

7.6 Declaraciones while

AmientrasLa declaración debe tener el siguiente formato:

mientras (*condición*) {
declaraciones ;
}

Un vacíomientrasLa declaración debe tener el siguiente formato:

mientras (*condición*) ;

7.7 Declaraciones do-while

Ahacer mientras La declaración debe tener el siguiente formato:

hacer {
declaraciones;
} mientras (*condición*) ;

7.8 Sentencias switch

AcambiarLa declaración debe tener el siguiente formato:

```
cambiar (  condición  ) {
caso ABC:
    declaraciones;
  /* se cae */ caso
  DEF:
    declaraciones ;
  romper;

caso XYZ:
    declaraciones ;
  romper;

por defecto:
    declaraciones ;
  romper;
}
```

Cada vez que un caso fracasa (no incluyebmireaAlaskadeclaración), agregue un comentario donde romperLa declaración normalmente sería. Esto se muestra en el ejemplo de código anterior con el /* se cae */ comentario.

CadacambiarLa declaración debe incluir un caso predeterminado.bTre hamiken el caso predeterminado es Es redundante, pero evita un error de caída si más tarde ocurre lo mismo.doelahsmimiSe añade r.

7.9 Sentencias try-catch

Aintentar atrapar La declaración debe tener el siguiente formato:

```
intentar {
    declaraciones;
} catch (ExceptionClass e) {
    declaraciones;
}
```

Aintentar atrapar La declaración también puede ir seguida de baleyaliado , que se ejecuta independientemente de si o noryEl bloque se ha completado correctamente.

```
intentar {
    declaraciones;
} catch (ExceptionClass e) {
    declaraciones;
} finalmente {
    declaraciones;
}
```

8 - Espacio en blanco

8.1 Líneas en blanco

Las líneas en blanco mejoran la legibilidad al separar secciones de código que están relacionadas lógicamente.

Siempre se deben utilizar dos líneas en blanco en las siguientes circunstancias:

• Entre secciones de un archivo fuente
• Entre definiciones de clase e interfaz

Siempre se debe utilizar una línea en blanco en las siguientes circunstancias:

• Entre métodos
• Entre las variables locales de un método y su primera declaración
• Antes de un comentario en bloque (ver sección 5.1.1) o de una sola línea (ver sección 5.1.2)
• Entre secciones lógicas dentro de un método para mejorar la legibilidad

8.2 Espacios en blanco

Se deben utilizar espacios en blanco en las siguientes circunstancias:

• Una palabra clave seguida de un paréntesis debe estar separada por un espacio. Ejemplo:

```
mientras (verdadero) {
    . . .
}
```

Tenga en cuenta que no se debe dejar un espacio en blanco entre el nombre del método y su paréntesis inicial. Esto ayuda a distinguir las palabras clave de las llamadas a métodos.

• Debe aparecer un espacio en blanco después de las comas en las listas de argumentos.

• Todos los operadores binarios, excepto t, deben separarse de sus operandos mediante espacios. Los espacios en blanco nunca deben separar los operadores unarios, como el restar, el incremento (“++”) y el decremento (“--”), de sus operandos. Ejemplo:

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

mientras (d++ = s++) {
    n++;
}
imprime("el tamaño es " + foo + "\n");
```

• Las expresiones enparaaoLas declaraciones deben estar separadas por espacios en blanco. Ejemplo:

```
para (expr1; expr2; expr3)
```

• Los moldes deben ir seguidos de un espacio en blanco. Ejemplos:

```
    miMétodo((byte) aNum, (Objeto) x);
miMétodo((int) (cp + 5), ((int) (i + 3))
                + 1);
```

9 - Convenciones de nomenclatura

Las convenciones de nomenclatura facilitan la comprensión de los programas al facilitar su lectura. También pueden proporcionar información sobre la función del identificador (por ejemplo, si se trata de una constante, un paquete o una clase), lo cual puede ser útil para comprender el código.

Tipo de identificador	Reglas para nombrar	Ejemplos
Paquetes	El prefijo de un nombre de paquete único siempre se escribe en letras ASCII minúsculas y debe ser uno de los nombres de dominio de nivel superior, actualmente com, edu, gov, mil, net, org, o uno de los códigos ingleses de dos letras que identifican países en desarrollo según lo especificado en la Norma ISO 3166, 1981. Los componentes posteriores del nombre del paquete varían según las convenciones de nomenclatura internas de la organización. Dichas convenciones pueden especificar que ciertos componentes del nombre del directorio sean nombres de división, departamento, proyecto, equipo o inicio de sesión.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.queso
Clases	Los nombres de las clases deben ser sustantivos, en mayúsculas y minúsculas. domimuchacha Raster; con la primera letra de cada palabra interna en mayúsculaclase ImageSprite; Talizado. Procura que los nombres de tus clases sean sencillos y descriptivos. Usa palabras completas; evita acrónimos y abreviaturas (a menos que la abreviatura se use mucho más que la forma larga, como URL o HTML).	
Interfaces	Los nombres de interfaz deben escribirse con mayúscula, al igual que los nombres de clase.	interfaz RasterDelegate; interfaz Almacenando;
Métodos	Los métodos deben ser verbos, en mayúsculas y minúsculas concorrer(); la primera letra minúscula, con la primera letra decorrerRápido(); Cada palabra interna en mayúscula.	obtenerFondo();

Tipo de identificador	Reglas para nombrar	Ejemplos
Variables	A excepción de las variables, todas las instancias, clases yentero Las constantes de clase están en mayúsculas y minúsculas.dooh - Arkansas mayúscula inicial. Las palabras internas empiezan con mayúscula. flotar Letras mayúsculas. Los nombres de las variables no deben comenzar con guion bajo _ ni con el signo de dólar $, aunque ambos estén permitidos. Los nombres de las variables deben ser breves pero significativos. La elección del nombre de una variable debe ser mnemotécnica, es decir, diseñada para indicar al observador casual la intención de su uso. Se deben evitar los nombres de variable de un solo carácter, excepto para variables temporales "desechables". Nombres comunes para variables temporales... irej, k, metro,ynortepara números enteros;cd,ymipara personajes.	i; do; miAncho;
Constantes	Los nombres de las variables declaradas por la clase con-Las constantes ANSI deben escribirse en mayúsculas, separadas por guiones bajos ("_"). (Para facilitar la depuración, se deben evitar las constantes ANSI).	int final estático MIN_WIDTH = 4; int final estático MAX_WIDTH = 999; int final estático OBTENER_LA_CPU = 1;

10 - Prácticas 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 configurarse ni obtenerse explícitamente; esto suele ocurrir como efecto secundario de las llamadas a métodos.

Un ejemplo de variables de instancia públicas apropiadas es el caso en que la clase es esencialmente una estructura de datos, sin comportamiento. En otras palabras, si tuviera que usartrdcael en lugar de una clase (si Java lo soporta)sdconstrucción) , entonces es apropiado hacer que las variables de instancia de la clase público.

10.2 Referencia a variables y métodos de clase

Evite usar un objeto para acceder a una variable o método de clase (estático). En su lugar, use el nombre de una clase. Por ejemplo:

método de clase(); //DE ACUERDO
Una clase.classMethod(); //DE ACUERDO

```
anObject.classMethod(); //¡EVITAR!
```

10.3 Constantes

Las constantes numéricas (literales) no deben codificarse directamente, excepto -1, 0 y 1, que pueden aparecer en unaparabucle como valores de contador.

10.4 Asignaciones de variables

Evite asignar varias variables al mismo valor en una sola declaración. Es difícil de leer. Ejemplo:

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

No utilice el operador de asignación donde pueda confundirse fácilmente con el operador de igualdad. Ejemplo:

```
si (c++ = d++) {          // ¡EVITAR! (Java no lo permite)
   ...
}
```

Debería escribirse como

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

No utilice asignaciones incrustadas para intentar mejorar el rendimiento en tiempo de ejecución. Esto es tarea del compilador. Ejemplo:

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

Debería escribirse como

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

10.5 Prácticas diversas

10.5.1 Paréntesis

Generalmente, es recomendable usar paréntesis con liberalidad en expresiones que involucran operadores mixtos para evitar problemas de precedencia. Aunque la precedencia de operadores le parezca clara, puede que no lo sea para otros; no asuma que otros programadores la conocen tan bien como usted.

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

si ((a == b) && (c == d)) // USAR
```

10.5.2 Devolviendo valores

Intenta que la estructura de tu programa coincida con la intención. Ejemplo:

```
si (expresión booleana ) {
    devuelve verdadero;
} demás {
    devuelve falso;
}
```

Debería escribirse en cambio como

```
devolver expresión booleana ;
```

Similarmente,

```
si (condición) {
    devuelve x;
}
devuelve y;
```

Debería escribirse como

```
devolver (condición ? x : y);
```

10.5.3 Expresiones antes de '?' en el operador condicional

Si una expresión que contiene un operador binario aparece antes de ?ein el operador ternario?: debe estar entre paréntesis. Ejemplo:

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

10.5.4 Comentarios especiales

UsarXXXen un comentario para marcar algo que es falso pero funciona.FTúIXmimimarcar algo Eso es falso y está roto.

11 - Ejemplos de código

11.1 Ejemplo de archivo fuente de Java

El siguiente ejemplo muestra cómo formatear un archivo fuente Java que contiene una sola clase pública. Las interfaces tienen un formato similar. Para más información, consulte “Declaraciones de clases e interfaces” en la página 3 y “Comentarios de la documentación” en la página 8.

```
/*
 * @(#)Blah.java 1.82 99/03/18
 *
 * Copyright (c) 1994-1999 Sun Microsystems, Inc.
 * 901 San Antonio Road, Palo Alto, California, 94303, EE. UU.
 * Reservados todos los derechos.
 *
 * Este software es información confidencial y propiedad de Sun
 * Microsystems, Inc. ("Información Confidencial"). No deberá
 * revelar dicha Información Confidencial y utilizarla únicamente en
 * de acuerdo con los términos del acuerdo de licencia que usted firmó
 * con el sol.
 * /


paquete java.blah;

importar java.blah.blahdy.BlahBlah;

/**
 *    La descripción de la clase va aquí              .
 *
 * @versión              1.82 18 Mar 1999
 * @autor                Nombre Apellido
 * /
clase pública Blah extiende SomeClass {
    /* Aquí puede ir un comentario de implementación de clase. */

    /**         Comentario de la documentación de classVar1                 * /
    público estático int claseVar1;

    /**
         *  Comentario de la documentación de classVar2 que tiene más
     *       de una línea de longitud
     * /
    clase de objeto estático privadoVar2;

    /**         Comentario de la documentación de instanceVar1                  * /
    Objeto público instanceVar1;

    /**         Comentario de la documentación de instanceVar2                  * /
    int protegido instanceVar2;

    /**         Comentario de la documentación de instanceVar3                  * /
    objeto privado[] instanceVar3;

    /**
     *...            constructor Blah documentación comentario...
     * /
    público Blah() {
        //...la implementación va aquí... }


    /**
     *...            método doSomething documentación comentario...
     * /
    público void hacerAlgo() {
       //...la implementación va aquí... }
```

```
    /**
     * . . . método hacerAlgoMás                    comentario de documentación...
     * @param algúnParámetro          descripción
     * /
    public void hacerAlgoMás(Objeto algúnParámetro) {
        //...la implementación va aquí... }

}
```