GUÍA DE ESTILO EN JAVA

Formato de líneas

1. No usar más de 80 caracteres por línea (imagen de tarjeta). De esta forma se pueden visualizar las líneas completas con un editor de texto o en una hoja impresa tamaño DIN A4.

2. Cuando la línea sea mayor de 80 caracteres, divídala en varias partes, cada una sobre una línea. Salte de línea al final de una coma o al final de un operador. Si se trata de una expresión con paréntesis salte, si es posible, a línea nueva después de finalizar el paréntesis. Por ejemplo, casos válidos serían,

```
public void metodoHaceAlgo(int valor1, double valor2,
                           int valor3){


resultado = aux* (final-inicial+desplazamiento)
                 + referencia;
```

3. Use líneas en blanco como elemento de separación entre bloques de código conceptualmente diferentes.

4. Sangre adecuadamente cada nuevo bloque de sentencias. Entre dos y cuatro espacios en blanco son suficientes para cada nivel de sangrado. De esta forma se aprecia visualmente la diferencia de nivel entre bloques de sentencias, sin rebasar, normalmente, los 80 caracteres por línea. A la hora de sangrar, y para evitar problemas de compatibilidad entre editores de texto, use espacios y no tabuladores. Por ejemplo,

```
public double calcularDescuento (double total) {

  int aux=0;    // Se sangra dos espacios

  if (total>LIMITE) {
    total = total * 0.9;  // Se sangra otros dos espacios
  }
  return total;
}
```

Ficheros

1. Incluya una sola clase o interfaz por fichero, o al menos una sola clase o interfaz pública.

2. Si hay comentarios globales para los contenidos del fichero colóquelos en primer lugar.

3. Si la clase forma parte de un paquete, la sentencia `package` deber ser la primera del fichero.

4. Si se importan paquetes, la sentencia `import` debe aparecer después de la sentencia `package`.

5. Coloque la declaración de las clases o interfaces a continuación de la sentencia `package`.

Clases

1. Coloque en primer lugar los comentarios sobre la clase (vea el apartado comentarios).

2. Coloque los atributos (datos) a continuación. Coloque primero las variables estáticas y a continuación las variables de ejemplar en el orden: público (se recomienda no incluir variables públicas), protegidas y privadas, es decir, `public`, `protected` y `private` en Java.

3. A continuación se declaran los métodos, con los constructores en primer lugar. El resto de métodos se colocará en orden de interrelación y no por visibilidad. Así, si un método público usa dos métodos privados, se debería colocar primero el público seguido de los dos privados. La idea es que al leer el código se siga con facilidad la funcionalidad de los métodos. Las recomendaciones de los dos últimos puntos se resumirían de la forma siguiente,

```
class NombreClase {
  variables estáticas
  variables de ejemplar públicas (debe evitarse su uso)
  variables de ejemplar protegidas
  variables de ejemplar privadas

  métodos constructores
  resto de métodos
}
```

4. No deje espacio en blanco entre el identificador del método y los paréntesis, es decir, escriba,

```
public void nombreMétodo(int valor1, double valor2)
```

en lugar de,

```
public void nombreMétodo (int valor1, double valor2)
```

obsérvese el espacio en blanco delante de la apertura de paréntesis.

Visibilidad de miembros de clase

1. Los atributos (datos) deben declararse privados (`private`) excepto los que se pretenda que sean accesibles por herencia que deben ser protegidos (`protected`). Se debe evitar el uso de datos públicos.

2. Los procedimientos (métodos) de la interfaz pública deben declararse `public`, los de soporte privados (`private`).

Identificadores

1. Escoja identificadores significativos y a ser posible breves. En cualquier caso prefiera la claridad a la brevedad. Por ejemplo, es preferible el identificador,

   ```
   integralIndefinida
   ```

   que el de,

   ```
   intInd
   ```

2. Para clases e interfaces escoja como identificador un sustantivo. Use minúsculas excepto para la letra inicial. Si el identificador consta de varias palabras colóquelas juntas, con la inicial de cada una en mayúsculas. Por ejemplo, serían identificadores apropiados,

   ```
   class Cliente

   class ClientePreferencial
   ```

3. Para las variables y objetos utilice identificadores en minúsculas. Si el identificador consta de varias palabras se colocan separadas por el carácter de subrayado o bien todas seguidas. Es este último caso, la primera de ellas se escribe en minúsculas y la inicial de las demás en mayúsculas. Por ejemplo,

   ```
   double valorFinal=0.0;
   int cantidadFinal=0;
   String nombre_cliente="Aurora";
   ```

4. Para las constantes, el identificador debe usarse en mayúsculas. Si el identificador consta de varias palabras se separan con el carácter de subrayado. Ejemplo correctos serían,

   ```
   final int MINIMO=100;
   final double PRECISION=0.001;
   final double ALTURA_MEDIA=29.5;
   ```

5. En el caso de los métodos, el identificador debe ser preferentemente un verbo y debe usarse en minúsculas. Si el identificador contiene varias palabras, la inicial de todas las posteriores a la primera va en mayúsculas. Ejemplos válidos serían,

   ```
   public void introducir(double valor){
     - - - cuerpo del método - - -
   }

   public double obtenerMedia(){
     - - - cuerpo del método - - -
   }
   ```

6. Para el identificador de los paquetes se recomienda usar minúsculas.

Declaraciones

1. Declare las variables al principio del bloque en el que se vayan a utilizar. En los métodos, declare las variables al principio del método. Intente inicializar todas las variables que se declaren.

2. En relación con el punto anterior, si en los bucles no se precisa que alguna variable exista antes o después del mismo, declárela dentro del bucle.

3. Declare las matrices con los corchetes al lado del nombre del tipo y no del identificador, es decir, con la sintaxis,

```
int [] producto;
```

y no como,

```
int producto [];
```

Sentencias de control

1. No use nunca saltos incondicionales. Más concretamente, no utilice la sentencia `break` excepto en la sentencia `switch`, donde es forzoso hacerlo. Nunca use la sentencia `continue`.

2. No use más de un `return` en cada método y coloque éste al final del método.

3. Coloque la apertura de bloque (`{`) al final de la línea inicial de la sentencia. El fin de bloque (`}`) colóquelo en línea aparte y alineado con el principio de la sentencia. Si la sentencia de control es un `if-else` la cláusula `else` debe comenzar en la línea siguiente al fin del bloque de la cláusula `if`. Por ejemplo, escriba,

```
if (i==j) {
  dato=5;
}
else {
  dato=6;
}
```

en lugar de,

```
if (i==j)
{
  dato=5;
} else
{
  dato=6;
}
```

4. Ponga cada nuevo bloque de sentencias entre llaves aunque conste de una sola sentencia. Por

ejemplo, escriba

```
if (i==j) {
  dato=5;
}
else {
  dato=6;
}
```

y no,

```
if (i==j)
  dato=5;
else
  dato=6;
```

5. Una recomendación muy frecuente es la siguiente. En los `if` anidados si un `if` interno corresponde a una cláusula `else` externa se coloca el `if` a continuación y en la misma línea que el `else`. El bloque de este nuevo `if` se cierra al nivel del `else` anterior. Por ejemplo,

```
if (dato1==0) {
  resultado=total;
} else if (dato1>10) {
  resultado = total*0.8;
} else {
  resultado = total*0.95;
}
```

Aunque ésta es una recomendación muy extendida que intenta que los `if` se interpreten en cascada, aquí recomendamos el formato obtenido al aplicar las normas del apartado 3 de esta sección, que para el ejemplo anterior produciría el siguiente resultado,

```
if (dato1 == 0){
  resultado=total;
}
else {
  if (dato1 > 10) {
    resultado = total*0.8;
  }
  else {
    resultado = total*0.95;
  }
}
```

Este formato permite identificar el alcance de las distintas estructuras `if-else` por el nivel de sangrado de las sentencias.

Documentación

1. Documente el código. No se trata de sustituir la información de análisis y diseño, pero la documentación interna es un complemento que puede servir como guía del sistema en el peor de los casos, cuando no hay otra documentación disponible.

2. Como cabecera de una clase incluya como información el autor o autores, la fecha de la última modificación, el propósito general de la clase y una breve explicación de los datos y métodos incorporados.

3. Como cabecera de los métodos cuya acción no sea evidente indique el autor o autores, la fecha de la última modificación, el propósito del método, el significado de los parámetros formales, el significado de la información devuelta con `return` y las excepciones que se capturen.

4. En el cuerpo de los métodos use comentarios para indicar el propósito de las tareas que no sean evidentes, tales como algoritmos específicos. En cualquier caso aumente la legibilidad del código usando identificadores significativos para variables o métodos.