Convenio de codificación general para todos los lenguajes

Las buenas prácticas de codificación ayudan a generar un código más eficiente, claro y fácil de mantener.

Es muy común que el estilo de programación dependa en gran medida del lenguaje de programación que se haya elegido para desarrollar la aplicación, pero hay una serie de convenciones válidas para cualquier tipo de lenguaje. A continuación se ofrece un conjunto de pautas para mejorar las buenas prácticas en la codificación.

Título	Carácter
Nomenclatura de las variables	**Recomendada**
Uso de la identación	**Recomendada**
Controlar el tamaño del código fuente	**Recomendada**
Uso de variables booleanas en las estructuras de decisión	**Obligatoria**
Uso de estructuras de control lógicas	**Obligatoria**
Manejo de los espacios blancos	**Recomendada**
Eliminar el uso de número mágicos	**Obligatoria**
Tamaño de los métodos	**Obligatoria**
Número de máximo de parámetros	**Obligatoria**
Búsqueda por cadenas Texto	**Obligatoria**
Encoding	**Obligatoria**

- **Nomenclatura de las variables**

  Las variables deben nombrarse de manera que facilite la comprensión del código fuente. Por ello deben de mantener una cierta lógica con el modelo de negocio que representan. De esta manera, el desarrollador obtiene un código más intuitivo y que se puede mantener.

- **Uso de la identación**

  Debe utilizarse un estilo de sangría, en lenguajes de programación que usan llaves para tabular o delimitar bloques lógicos de código. Usando un estilo lógico y consistente hace que el código realizado sea más legible.

- **Controlar el tamaño del código fuente**

  Se debe de controlar el tamaño del código fuente estructurado. No debe sobrepasarse un determinado número de líneas de código porque implicaría probablemente un mal diseño. En el caso de Java, no deben de superarse las dos mil líneas de código.

- **Uso de variables booleanas en las estructuras de decisión**

  En estructuras de decisión se recomienda el uso de variables booleanas, ya que la no utilización de estas variables provoca que el código sea más complejo y difícil de entender, provocando incluso algunos errores en su concepción.

```
/* Un buen ejemplo
return horas < 12 && minutos < 60 && segundos < 60;
```

- **Uso de estructuras de control lógicas**

  El uso de estructuras de control lógicas para bucles también es parte de un buen estilo de programación. Ayuda a quien esté leyendo el código a entender la secuencia de ejecución (en programación imperativa). Por ejemplo, el siguiente pseudocódigo:

```
cuenta = 0
 while cuenta < 5
   print cuenta * 2
   cuenta = cuenta + 1
 end while
```

  El extracto anterior cumple con las recomendaciones de estilo anteriores, pero el siguiente uso de la construcción "for" hace el código mucho más fácil de leer:

```
for (cuenta = 0, cuenta < 5, cuenta=cuenta+1) {
  System.out.println(cuenta * 2);
}
```

- **Manejo de los espacios blancos**

  Los lenguajes de formato libre ignoran frecuentemente los espacios en blanco. El buen uso del espaciado en la disposición del código es, por tanto, considerado un buen estilo de programación.

- **Eliminar el uso de número mágicos**

  En ocasiones, se tiende a introducir números para realizar acciones condicionales o comprobaciones en el código. Es mucho más recomendable el uso de constantes para estas situaciones, de manera que el costo del mantenimiento del código se reduce ante cualquier modificación que afecte a la estructura.

```
// Una mala estructura sería la siguiente
if (lineas <= 100){
.
.
}

// Una buena estructura
int TAMAÑO=100;
if (lineas <= TAMAÑO){
.
.
}
```

- **Tamaño de los métodos**

  La longitud de un método no debe de exceder de cien líneas de código sin una causa justificada.

  En ocasiones, se intenta agrupar excesiva funcionalidad dentro de un mismo método. Esto provoca que la detección de un error sea compleja, dificultando la legibilidad y mantenimiento del código.

- **Número de máximo de parámetros**

  Se recomienda no exceder de 10 parámetros en las llamadas a métodos si no existe una causa justificada para ello. De esta manera se asegura que los métodos no se encuentran sobrecargados de funcionalidad, favoreciendo el mantenimiento eficiente del código.

- **Búsqueda por cadenas Texto**

  Cuando se elaboren búsqueda por cadenas de texto no se debe hacer discriminación por mayúsculas, minúsculas o acentuación a la hora de obtener los registros asociados a la búsqueda.

- **Encoding**

  Todos los ficheros entregables deben de ser codificados en UTF-8.

  Se adoptará el método de codificación UTF-8 para el código fuente en lugar de ISO-8859-1 (por defecto en MS Windows, no así en Linux o Mac OS). Ello es coherente con la corriente actual de desarrollo, facilita la portabilidad de los sistemas de información entre diversos sistemas operativos, y evitará problemas posteriores a la hora de generar documentación desde el código fuente, como el Javadoc o el código fuente navegable (HTML de nuestro código, gracias al plugin JXR de Maven).

  Actualmente, todos los protocolos de internet y todos los programas de correos electrónicos son capaces de interpretar y mostrar mensajes codificados mediante UTF-8. Para estandarizar el desarrollo, deberá configurarse el editor de texto IDE utilizado para que almacene los ficheros de código fuente en UTF-8. Esta operación sólo es necesaria en Windows ya que Linux trabaja por defecto con UTF-8.