JavaDoc. Comentando el código fuente.

Este artículo es el inicio de una serie de ellos dedicados a aprender cómo comentar el código Java y cómo generar la documentación HTML de nuestras API's mediante la herramienta de generación automática **Javadoc**.

Como mi mentor en Java me dijo una vez, el buen código en Java se comenta solo y no le falta razón. Por eso, aquí vamos a aprender que no es necesario comentar todas las líneas de código (salvo algún caso excepcional que requiera ser remarcado por su importancia). Realmente lo único necesario es comentar la clase y cada uno de los métodos que empleemos, bien para quien quiera entender qué hace esa clase o la función que desenpeñan cada uno de los métodos o bien, como es la intención de este artículo para que luego la herramienta de generación automática de documentación nos cree la nuestra.

En este primer artículo nos dedicaremos a aprender cómo se comentan las clases y las etiquetas que disponemos para añadir información adicional.

Tipo de comentario para generar la documentación.

A parte de los ya sabidos /* */ ó //, también tenemos un tercero propio de java y que es el que usa el javadoc como delimitador para reconocer la documentación que tiene que generar. Cualquier comentario de este tipo debe comenzar por el delimitador **/**** y, luego, cada línea de comentario que añadamos debe seguir al operador *****. Así mismo, también comenzarán por un * las etiquetas, sólo que éstas a su vez deben ser precedidas por el operador @ como se verá en el próximo
capítulo.

Una cosa muy importante que quiero resaltar ya es que javadoc lo único que hace es copiar el contenido a una página HTML por lo que se admite cualquier etiqueta HTML que queráis para darle más claridad o vistosidad a vuestra documentación. Vemos a continuación el formato típico de un comentario al estilo javadoc.

```
/**
* Al usar este estilo de comentario estoy consiguiendo que cuando
* lo pase por javadoc, esta herramienta copie este texto a una página
* html.
* Puedes ver que  también admite <b>etiquetas HTML</b>
* y ésto último que acabo de escribir aparecerá en negrita.
*/
```

Las etiquetas.

Aparte de los comentarios propios, la utilidad javadoc nos proporciona una serie de etiquetas para completar la información que queremos dar de una determinada clase o método. Son las siguientes:

@author *nombre* **(desde la versión 1.0 del JDK/SDK)**

Indica el autor de la clase en el argumento *nombre*. Un comentario de este tipo puede tener más de un autor en cuyo caso podemos usar tantas etiquetas de este tipo como autores hayan colaborado en la creación del código fuente o bien podemos ponerlos a todos en una sola etiqueta. En éste último caso, Javadoc inserta una (,) y un espacio entre los diferentes nombres.

@deprecated *comentario* **(desde la versión 1.0 del JDK/SDK)**

Añade un *comentario* indicando que este API no debería volver a usarse, aunque aun siga perteneciendo a la distribución del SDK que estemos utilizando, por estar desautorizado o "desfasado". No obstante, ésto es sólo una advertencia que nosotros damos a los usuarios de nuestras API's, al igual que las distribuciones de Sun hacen con nosotros. Realmente, lo que estamos haciendo al decir que una determinada API está desfasada es prevenir de que en un futuro podrán surgir incompatibilidades si seguimos usándolas ya que éste es el paso a previo a la desaparición del API en concreto.

En la primera frase del comentario, que es la que la documentación nos la muestra en la sección del resumen de nuestra clase, deberíamos como mínimo poner desde que versión de nuestra API está desautorizada y por quién se debería sustituir. A partir de Java 1.2 podemos usar `{@link}` para referenciar por quién debemos hacer la sustitución.

@exception *nombre-clase descripción* **(desde la versión 1.0 del JDK/SDK)**

Esta etiqueta actúa exactamente igual que `@throws`.

{@link *nombre etiqueta***} (desde la versión 1.2 del JDK/SDK)**

Inserta un enlace autocontenido que apunta a *nombre.* Esta etiqueta acepta exactamente la misma sintáxis que la etiqueta `@see`, que se describe más abajo, pero genera un enlace autocontenido en vez de colocar el enlace en la sección "See Also". Dado que esta etiqueta usa los carácteres { } para separarla del resto del texto in-line, si necesitas emplear el caracter "}" dentro de la etiqueta debes usar la notación HTML }

No existe ninguna limitación en cuanto al número de etiquetas de este tipo permitidas. Puedes usarlas tanto en la parte de la descripción como en cualquier porción de texto de cualquier otra etiqueta de las que nos proporciona JavaDoc.

En el siguiente ejemplo, vemos como crear dentro de nuestra documentación un enlace in-line al método getComponentAt(int, int).

@deprecated Desde JDK1.1 Usar el método `{@link #getComponentAt(int, int) getComponentAt}`

A partir de esta descripción, la herramienta de generación automática de código generará el siguiente código HTML (supone que se está refiriendo a una clase del mismo paquete):

`Usar el método <a href="Component.html#getComponentAt(int, int)">getComponentAt</a>` , el cual aparecerá en la página HTML como:

Usar el método getComponentAt

@param *parámetro descripción* **(desde la versión 1.0 del JDK/SDK)**

Añade un parámetro y su descripción a la sección "Parameters" de la documentación HTML que generará. Por tanto, para cada método emplearemos tantas etiquetas de este estilo como parámetros de entrada tenga dicho método.

@return *descripción* **(desde la versión 1.0 del JDK/SDK)**

Añade a la sección "Returns" de la documentación HTML que va a generar la descripción del tipo que devuelve el método. Veámos el siguiente ejemplo:

Si suponemos que tenemos el método `public String concatena(String s1, String s2, String s3)` deberíamos comentar nuestro código fuente de la siguiente manera:

```
/*
* ....
* @param s1 Texto que ocupa la cabecera
* @param s2 Texto que ocupa el cuerpo de la string a crear
* @param s3 Texto que ocupa la parte final
* @return La cadena conformada de tipo String.
*/
```

@see *referencia* **(desde la versión 1.0 del JDK/SDK)**

Añade un cabecero "See Also" con un enlace o texto que apunta a una referencia. El comentario de la documentación puede contener cualquier número de etiquetas de este tipo y todas, al generar la documentación, se agruparán bajo el mismo cabecero. Esta etiqueta se puede conformar de tres maneras diferentes, siendo la última forma la más utilizada.

`@see "`*`string`*`"`

En este caso, no se genera ningún enlace. La *string* es un libro o cualquier otra referencia a una información que no está disponible via URL. JavaDoc distingue este caso buscando en el primer carácter de la cadena las comillas dobles ("). Por ejemplo:

```
@see "The Java Programming Language"
```

que genera el siguiente texto HTML:

See Also:

"The Java Programming Language"

`@see <a href="`*`URL#value`*`">`*`label`*`</a>`

Añade un enlace definido por URL#value. Esta dirección puede ser relativa o absoluta. JavaDoc distingue este caso buscando en el primer carácter el símbolo "<". Por ejemplo:

```
@see <a href="spec.html#section">Java Spec</a>
```

que genera el siguiente enlace:

See Also:

Java Spec

@see *package.class#member texto*

Añade un enlace, con el texto visible *texto*, que apunta en la documentación al nombre especificado en el lenguaje Java al cual hace referencia. Aquí por nombre se debe entender un paquete, una clase, un método o un campo. El argumento *texto* es opcional, si se omite, JavaDoc nos dará la información mínima, ésto es, el nombre de, por ejemplo, la pareja `paquete#método`. Usa este campo cuando quieras especificar algo más, cuando quieras representarlo por un nombre más corto o simplemente si quieres que aparezca con otro nombre. Veámos a continuación varios ejemplos usando la etiqueta `@see`.

Nota: El comentario a la derecha muestra cómo aparecerá la etiqueta en las especificaciones HTML si la referencia a la que apunta estuviera en otro paquete distinto al que estamos comentando.

```
                                        See also:
@see java.lang.String                  //
String
@see java.lang.String The String class  //  The String
class
@see String                            //
String
@see String#equals(Object)             //
String.equals(Object)
@see String#equals                     //
String.equals(java.lang.Object)
@see java.lang.Object#wait(long)       //
java.lang.Object.wait(long)
@see Character#MAX_RADIX               //
Character.MAX_RADIX
@see <a href="spec.html">Java Spec</a>  //  Java Spec
@see "The Java Programming Language"   //  "The Java Programming
Language"
```

@since *texto* **(desde la versión 1.1 del JDK/SDK)**

Indica con *texto* desde cuándo se creó este paquete, clase o método. Normalmente se pone la versión de nuestra API en que se incluyó, así en posteriores versiones sabremos a qué revisión pertenece o en qué revisión se añadió. Por ejemplo:

```
@since JDK1.1
```

@serial *field-description* **(desde la versión 1.2 del JDK/SDK)**

Su uso está destinado a señalar un campo serializable. Por defecto, todos los campos (variables) son susceptibles de ser serializados lo cual no quiere decir que nuestra aplicación lo tenga que hacer. Por tanto, úsala sólo cuando tengas una variable serializable. Aprovecho para decirte que no te asustes si tu clase tiene, digamos, 20 variables y al generar la documentación HTML te "cantan" 20 warnings. Ésto ocurre porque JavaDoc esperaba que hubieras asignado una etiqueta `serial` a cada una de tus variables ya que como dije antes, en principio todas estas variables pueden ser serializables. En fin, la 2ª vez que generes la documentación ya no te asustará ;-).

@serialField *field-name field-typefield-description* **(desde la versión 1.2 del JDK/SDK)**

Documenta un componente `ObjectStreamField` de un miembro `serialPersistentFields` de una clase `Serializable`. Esta etiqueta se debería usar para cada componente `ObjectStreamField`.

@serialData *data-description* **(desde la versión 1.2 del JDK/SDK)**

Se emplea para describir los datos escritos por el método `writeObject` y todos los datos escritos por el método `Externalizable.writeExternal`. Esta etiqueta puede ser usada en aquellas clases o métodos que intervengan los métodos `writeObject`, `readObject`, `writeExternal`, and `readExternal`.

@throws *nombre-clase descripcion* **(desde la versión 1.2 del JDK/SDK)**

Como ya se apuntó, esta etiqueta es la gemela de `@exception`. En ambos casos, se añade una cabecera "Throws" a la documentación generada con el nombre de la excepción que puede ser lanzada por el método (*nombre-clase*) y una descripción de por qué se lanza.

@version *version* **(desde la versión 1.0 del JDK/SDK)**

Añade un cabecero a la documentación generada con la versión de esta clase. Por versión, normalmente nos referimos a la versión del software que contiene esta clase o miembro.

Donde se emplean las etiquetas.

Una vez sabemos de las etiquetas que disponemos, pasamos a ver dónde se pueden emplear. Un detalle importante a tener en cuenta es que **SIEMPRE** que se quiera comentar algo, una clase, un método, una variable, etc..., dicho comentario se debe poner inmediatamente antes del item a comentar. En caso contrario la herramienta de generación automática no lo reconocerá.

Comentario de la página "Overview".

Las etiquetas que se pueden emplear en esta sección son: `@see, {@link}, @since`. Observa que ésto no se corresponde con ninguna parte de un código Java. La página "Overview", que reside en un fichero fuente al que normalmente se le da el nombre de `overview.html` es la que nos da una idea global de las clases y paquetes que conforman nuestra API (ver especificaciones html de Sun).

Comentario de paquetes.

Este tipo de comentarios tampoco lo hacemos en nuestro código fuente sino en una página html que reside en cada uno de los directorios de nuestros paquetes y que siempre se le da el nombre de `package.html`.

Las etiquetas que podemos utilizar en este caso son: `@see, {@link}, @since, @deprecated`

Comentario de clases e interfaces.

Disponemos de las siguientes etiquetas: `@see, {@link}, @since, @deprecated, @author, @version.` Veámos un ejemplo del comentario de una clase:

```
/**
* A class representing a window on the screen.
* For example:
* <pre>
*       Window win = new Window(parent);
*       win.show();
* </pre>
*
* @author  Sami Shaio
* @version %I%, %G%
* @see     java.awt.BaseWindow
* @see     java.awt.Button
*/
class Window extends BaseWindow {
   ...
}
```

Comentario de variables.

Las etiquetas que podemos utilizar en esta ocasión son: `@see, {@link}, @since, @deprecated, @serial, @serialField`. Un ejemplo:

```
/**
* The X-coordinate of the component.
*
* @see #getLocation()
*/
int x = 1263732;
```

Comentario de métodos y constructores.

En este caso disponemos de: `@see, {@link}, @since, @deprecated, @param, @return, @throws (@exception), @serialData`. Un ejemplo:

```
/**
* Returns the character at the specified index. An index
* ranges from <code>0</code> to <code>length() - 1</code>.
*
* @param     index  the index of the desired character.
* @return    the desired character.
* @exception StringIndexOutOfRangeException if the index is not in the
range <code>0</code>
* to <code>length()-1</code>.
* @see       java.lang.Character#charValue()
*/
public char charAt(int index) {
   ...
```

```
}
```

Próximo capítulo.

En el siguiente capítulo analizaremos las múltiples opciones o argumentos que la herramienta JavaDoc nos proporciona para generar la documentación HTML de nuestras API's.

Leo Suarez está actualmente realizando el Proyecto Fin de Carrera en Java para obtener el título de Ingeniero Superior de Telecomunicaciones por la Universidad de Las Palmas de Gran Canaria.
Cuando no está proyectando o escribiendo para javaHispano aprovecha para disfrutar con la novia y los amigos del estupendo clima de esa maravillosa isla.
Para cualquier duda o tirón de orejas, e-mail a: leo@javahispano.com