

8.B. Documentación del código.

Sitio: [Formación Profesional a Distancia](#)

Curso: Programación

Libro: 8.B. Documentación del código.

Imprimido por: Iván Jiménez Utiel

Día: lunes, 10 de febrero de 2020, 15:59

Tabla de contenidos

[1. Documentación del código.](#)

[1.1. Etiquetas y posición.](#)

[1.2. Uso de las etiquetas.](#)

[1.3. Orden de las etiquetas.](#)

1. Documentación del código.

Llegados a este punto, vamos a considerar una cuestión de gran importancia, la documentación del código fuente. Piensa en las siguientes cuestiones:

- ¿Quién crees que accederá a la documentación del código fuente? Pues serán los autores del propio código u otros desarrolladores.
- ¿Por qué hemos de documentar nuestro código? Porque facilitaremos su mantenimiento y reutilización.
- ¿Qué debemos documentar? Obligatoriamente: clases, paquetes, constructores, métodos y atributos.
Opcionalmente: bucles, partes de algoritmos que estimemos oportuno comentar, ...

A lo largo de nuestra vida como programadores es probable que nos veamos en la necesidad de reutilizar, modificar y mantener nuestro propio código o incluso, código de otros desarrolladores. ¿No crees que sería muy útil que dicho código estuviera convenientemente documentado? ¿Cuántas veces no hemos leído código de otros programadores y quizá no hayamos comprendido qué estaban haciendo en tal o cual [método](#)? Como podrás comprender, la generación de una documentación adecuada de nuestros programas puede suponer una inestimable ayuda para realizar ciertos procesos en el software.

Si analizamos la documentación de las clases proporcionada en los paquetes que distribuye Sun, nos daremos cuenta de que dicha documentación ha sido generada con una herramienta llamada **Javadoc**. Pues bien, nosotros también podremos generar la documentación de nuestro código a través de dicha herramienta.

Si desde el principio nos acostumbramos a documentar el funcionamiento de nuestras clases desde el propio código fuente, estaremos facilitando la generación de la futura documentación de nuestras aplicaciones. ¿Cómo lo logramos? A través de una serie de comentarios especiales, llamados **comentarios de documentación** que serán tomados por **Javadoc** para generar una serie de archivos HTML que permitirán posteriormente, navegar por nuestra documentación con cualquier navegador web.

Los comentarios de documentación tienen una marca de comienzo (**/****) y una marca de fin (***/**). En su interior podremos encontrar dos partes diferenciadas: una para realizar una descripción y otra en la que encontraremos más etiquetas de documentación. Veamos un ejemplo:

```
/**
 * Descripción principal (texto/HTML)
 *
 * Etiquetas (texto/HTML)
 */
```

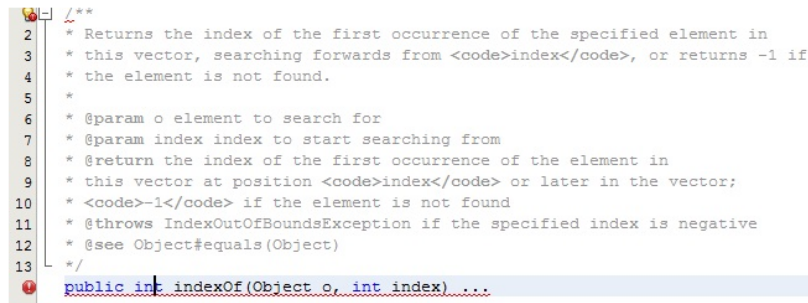
Este es el formato general de un comentario de documentación. Comenzamos con la marca de comienzo en una línea. Cada línea de comentario comenzará con un asterisco. El final del comentario de documentación deberá incorporar la marca de fin. Las dos partes diferenciadas de este comentario son:

- **Zona de descripción:** es aquella en la que el programador escribe un comentario sobre la [clase](#), [atributo](#), constructor o [método](#) que se vaya a [codificar](#) bajo el comentario. Se puede incluir la cantidad de texto que se

necesite, pudiendo añadir etiquetas HTML que formateen el texto escrito y así ofrecer una visualización mejorada al generar la documentación mediante **Javadoc**.

- **Zona de etiquetas:** en esta parte se colocará un conjunto de etiquetas de documentación a las que se asocian textos. Cada etiqueta tendrá un significado especial y aparecerán en lugares determinados de la documentación, una vez haya sido generada.

En la siguiente imagen puedes observar un ejemplo de un comentario de documentación.



```
1  /**
2   * Returns the index of the first occurrence of the specified element in
3   * this vector, searching forwards from <code>index</code>, or returns -1 if
4   * the element is not found.
5   *
6   * @param o element to search for
7   * @param index index to start searching from
8   * @return the index of the first occurrence of the element in
9   * this vector at position <code>index</code> or later in the vector;
10  * <code>-1</code> if the element is not found
11  * @throws IndexOutOfBoundsException if the specified index is negative
12  * @see Object#equals(Object)
13  */
14  public int indexOf(Object o, int index) ...
```

Reflexiona

Documentar el código de un programa es añadir suficiente información como para explicar lo que hace, punto por punto, de forma que no sólo los ordenadores sepan qué hacer, sino que además los humanos entiendan qué están haciendo y por qué. Documentar un programa no es sólo un acto de buen hacer del programador por aquello de dejar la obra rematada. Es además una necesidad que sólo se aprecia en su debida magnitud cuando hay errores que reparar o hay que extender el programa con nuevas capacidades o adaptarlo a un nuevo escenario.

1.1. Etiquetas y posición.

Cuando hemos de incorporar determinadas etiquetas a nuestros comentarios de documentación es necesario conocer dónde y qué etiquetas colocar, según el tipo de código que estemos documentando en ese momento. Existirán dos tipos generales de etiquetas:

1. **Etiquetas de bloque:** Son etiquetas que sólo pueden ser incluidas en el bloque de documentación, después de la descripción principal y comienzan con el símbolo @.
2. **Etiquetas en texto:** Son etiquetas que pueden ponerse en cualquier punto de la descripción o en cualquier punto de la documentación asociada a una etiqueta de bloque. Son etiquetas definidas entre llaves, de la siguiente forma `{@etiqueta}`

En la siguiente tabla podrás encontrar una referencia sobre las diferentes etiquetas y su ámbito de uso.

	@autor	{@code}	{@docRoot}	@deprecated	@exception	{@inheritDoc}	{@link}	{@literal}
Descripción	✓		✓	✓			✓	✓
Paquete	✓		✓	✓			✓	✓
Clases e Interfaces	✓		✓	✓			✓	✓
Atributos			✓	✓			✓	✓
Constructores y métodos			✓	✓	✓	✓	✓	

1.2. Uso de las etiquetas.

¿Cuáles son las etiquetas típicas y su significado? Pasaremos seguidamente a enumerar y describir la función de las etiquetas más habituales a la hora de generar comentarios de documentación.



- **@autor texto con el nombre**: Esta etiqueta sólo se admite en clases e interfaces. El texto después de la etiqueta no necesitará un formato especial. Podremos incluir tantas etiquetas de este tipo como necesitemos.
- **@version texto de la versión**: El texto de la versión no necesitará un formato especial. Es conveniente incluir el número de la versión y la fecha de ésta. Podremos incluir varias etiquetas de este tipo una detrás de otra.
- **@deprecated texto**: Indica que no debería utilizarse, indicando en el texto las causas de ello. Se puede utilizar en todos los apartados de la documentación. Si se ha realizado una sustitución debería indicarse qué utilizar en su lugar. Por ejemplo:

```
@deprecated El método no funciona correctamente. Se recomienda el uso de {@ling  
metodoCorrecto}
```

- **@exception nombre-excepción texto**: Esta etiqueta es equivalente a @throws.
- **@param nombre-atributo texto**: Esta etiqueta es aplicable a parámetros de constructores y métodos. Describe los parámetros del constructor o método. Nombre-atributo es idéntico al nombre del parámetro. Cada etiqueta @param irá seguida del nombre del parámetro y después de una descripción de éste. Por ejemplo:

```
@param fromIndex: El índice del primer elemento que debe ser eliminado.
```

- **@return texto**: Esta etiqueta se puede omitir en los métodos que devuelven void. Deberá aparecer en todos los métodos, dejando explícito qué tipo o clase de valor devuelve y sus posibles rangos de valores. Veamos un ejemplo:

```
/**
```

```

* Chequea si un vector no contiene elementos.
*
* @return <code>verdadero</code>si solo si este vector no contiene componentes, esto
es, su tamaño es cero;
* <code>falso</code> en cualquier otro caso.
*/
public boolean VectorVacio() {
return elementCount == 0;
}

```

- **@see referencia:** Se aplica a clases, interfaces, constructores, métodos, atributos y paquetes. Añade enlaces de referencia a otras partes de la documentación. Podremos añadir a la etiqueta: cadenas de caracteres, enlaces HTML a páginas y a otras zonas del código. Por ejemplo:

```

* @see "Diseño de patrones: La reusabilidad de los elementos de la programación
orientada a objetos"
* @see <a href="http://www.w3.org/WAI/">Web Accessibility Initiative</a>
* @see String#equals(Object) equals

```

- **@throws nombre-excepción texto:** En nombre-excepción tendremos que indicar el nombre completo de ésta. Podremos añadir una etiqueta por cada excepción que se lance explícitamente con una cláusula **throws**, pero siguiendo el orden alfabético. Esta etiquetas es aplicable a constructores y métodos, describiendo las posibles excepciones del constructor/método.

Autoevaluación

¿Qué etiqueta podría omitirse en un método que devuelve **void**?

- ☐ @param.
☐ @throws.
☐ @return.

1.3. Orden de las etiquetas.

Las etiquetas deben disponerse en un orden determinado. Ese orden es el siguiente:

Orden de etiquetas de comentarios de documentación.	
Etiqueta.	Descripción.
@autor	En clases e interfaces. Se pueden poner varios. Es mejor ponerlas en orden cronológico.
@version	En clases e interfaces.
@param	En métodos y constructores. Se colocarán tantos como parámetros tenga el constructor o método . Mejor en el mismo orden en el que se encuentren declarados.
@return	En métodos.
@exception n	En constructores y métodos. Mejor en el mismo orden en el que se han declarado, o en orden alfabético.
@throws	Es equivalente a @exception .
@see	Podemos poner varios. Comenzaremos por los más generales y después los más específicos.
@deprecated	

Para saber más

Si quieres conocer cómo obtener a través de **Javadoc** la documentación de tus aplicaciones, sigue los siguientes enlaces:

[Documentación con Javadoc.](#)

[Documentación de clases y métodos con Javadoc.](#)