


# *T4* **DOCUMENTACIÓN**

Entornos de desarrollo  
CFGS DAW

Autores:  
Paco Aldarias  
Cristina Álvarez

Revisado por:  
Sergio Badal  
Javier Valero  
M.Carmen Safont

Curso 2023/2024

Licencia  **Reconocimiento – NoComercial – CompartirIgual (by-nc-sa):**  
No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

## Índice

1	DOCUMENTAR UN PROYECTO .....	3
2	TIPOS DE DOCUMENTACIÓN .....	3
2.1.	<i>Documentación interna</i> .....	6
2.1.1.	Comentarios en el código .....	6
2.1.2.	Prólogo en el código.....	6
2.1.3.	Declaración de datos .....	7
2.1.4.	Construcción de sentencias .....	7
2.1.5.	Entrada/salida.....	7
3	UNA HERRAMIENTA: JAVADOC .....	9
3.1.	<i>Cómo hacer comentarios javadoc</i> .....	9
3.2.	<i>Uso de etiquetas javadoc</i> .....	10
3.3.	<i>Ejemplo de etiquetado</i> .....	17
3.4.	<i>Javadoc en Netbeans</i> .....	19
3.4.1.	Cómo visualizar el API de las librerías estándar .....	19
3.4.2.	Cómo comentar un método .....	19
3.4.3.	Cómo añadir tags a un método .....	19
3.4.4.	Cómo generar un javadoc .....	20
3.4.5.	Cómo personalizar el formato de javadoc .....	21
3.4.6.	Cómo buscar métodos sin documentación .....	22
3.5.	<i>Javadoc en Eclipse</i> .....	23
4.	BIBLIOGRAFÍA Y ENLACES.....	27

## 1. DOCUMENTAR UN PROYECTO

Cuando desarrollamos un proyecto se debe **aportar su documentación**. Esto es, todo lo que hayamos necesitado para desarrollar el software (código, diagramas, manuales de uso...).

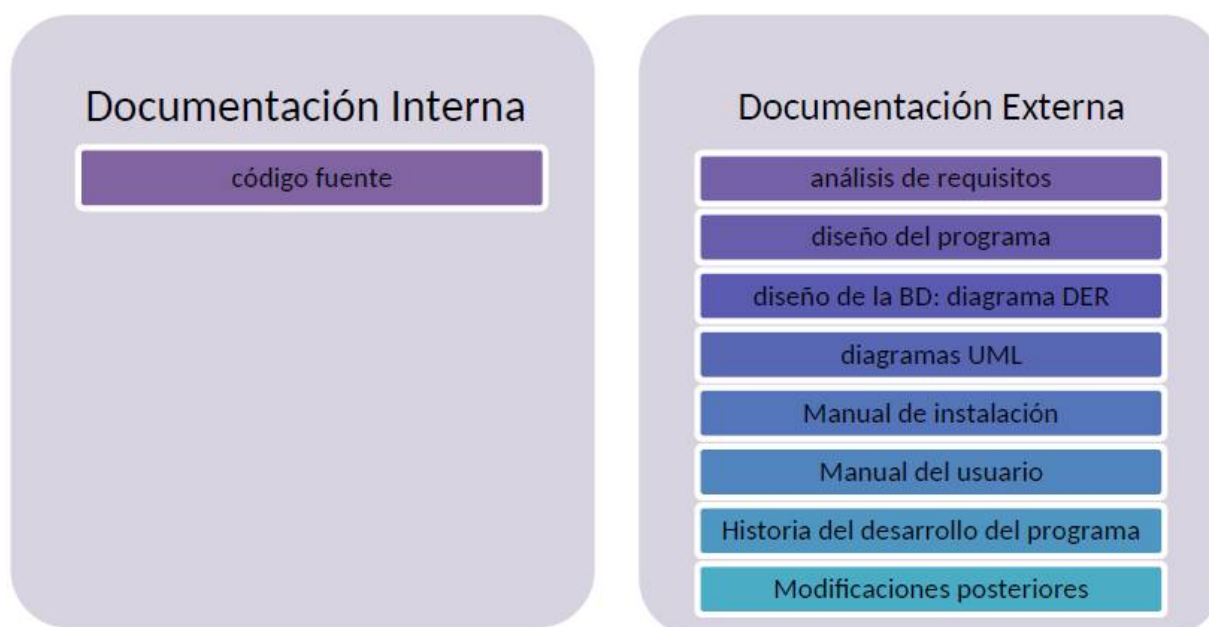
La **buena documentación** es esencial para un proyecto software. Sin ella un equipo se perderá en un mar de código. Por otro parte, demasiada documentación distrae e induce error.

Así, debe **crearse documentación** pero **prudentemente**. La documentación de software debe ser **corta y concreta**, explicando la esencia de cada elemento. Se puede poner esta documentación en un wiki, o alguna herramienta de colaboración para que cualquiera del equipo pueda acceder para poner su pantalla y buscarlas, y cualquiera pueda cambiarla cuando sea necesario. **Javadoc** es ejemplo de **herramienta de Java** para **escribir documentación**.

Existe una **metodología de planificación, desarrollo y mantenimiento** de sistemas de información promovida por el **Ministerio de Administraciones Públicas del Gobierno de España**. Su objetivo es la sistematización de actividades del ciclo de vida de los proyectos software en el ámbito de las administraciones públicas. Dicha metodología se conoce como **METRICA** y está **orientada al proceso**.

## 2. TIPOS DE DOCUMENTACIÓN

La información a entregar se clasifica en dos tipos, **interna** y **externa**, y es la siguiente:



El índice del documento a aportar una vez finalizado nuestro proyecto debería ser como sigue:

### **1.- Alcance Del Sistema**

- 1.- Planteamiento Del Problema
- 2.- Justificación
- 3.- Objetivos Generales Y Específicos
- 4.- Desarrollo Con Proceso Unificado

### **2.- Análisis Y Especificación De Requisitos**

- 1.- Identificación Y Descripción De Pasos
- 2.- Especificación De Requisitos
  - 1.- Objetivos Del Sistema
  - 2.- Requisitos De Información
  - 3.- Restricciones Del Sistema

### **3.- Requisitos Funcionales**

- 1.- Diagramas De Casos De Uso
- 2.- Definición De Actores
- 3.- Documentación De Los Casos De Uso
- 4.- Requisitos No Funcionales

### **3.- Diseño Del Sistema Xxx**

- 1.- Diagrama De Clases
- 2.- Modelo Entidad Relación
- 3.- Modelo Relacional
- 4.- Diccionario De Datos
- 5.- Diagrama De Secuencias
- 6.- Diagrama de actividades

### **4.- Implementación**

- 1.- Arquitectura Del Sistema
- 2.- Implementación Con Estándares
- 3.- Arquitectura De Desarrollo
- 4.- Estándar De Codificación
- 5.- Sistema De Control De Versiones
- 6.- Diagrama De Despliegue

### **5.- Pruebas**

- 1.- Planificación
- 2.- Desarrollo De Las Pruebas

### **6.- Resultados**

- 1.- Conclusiones
- 2.- Trabajos Futuros
- 3.- Anexos
  - 1.- Manual De Usuario
  - 2.- Manual De Instalación

(Fuente: Aldarias, 2012)

## 2.1. Documentación interna

Una vez que se genera el código fuente, la función de un módulo debe resultar clara sin necesidad de referirse a ninguna especificación del diseño. En otras palabras, el código debe ser comprensible.

El buen código se comenta solo, esto quiere decir que no es necesario comentar todas las líneas de código. Sin embargo, sí es necesario comentar algunas.

Vamos a distinguir entre comentarios dentro del código, prólogo, declaración de datos, construcción de sentencias y entrada/salida.

### 2.1.1. Comentarios en el código

La documentación interna del código comienza con la elección de los nombres (variables y etiquetas), continúa con la localización y composición de los comentarios y termina con la organización visual del programa.

La elección de nombres de identificadores significativos es crucial para la legibilidad. Los lenguajes que limitan la longitud de los nombres de las variables o de las etiquetas a unos pocos caracteres, implícitamente limitan la comprensión.

La posibilidad de expresar comentarios en lenguaje natural como parte del listado del código fuente es algo que aparece en todos los lenguajes de propósito general. Los comentarios pueden resultar una clara guía durante la última fase de la ingeniería del software, el mantenimiento.

### 2.1.2. Prólogo en el código

Al principio de cada módulo debe haber un comentario de prólogo que indique el nombre de la aplicación/módulo/fichero, resuma su función y describa su autor. Se propone la siguiente estructura:

1. Una sentencia que indique la función del módulo.
2. Una descripción de la interfaz :
  - a. un ejemplo de “secuencia de llamada”
  - b. una descripción de todos los argumentos
  - c. una lista de los módulos subordinados
3. Una explicación de los datos pertinentes, tales como las variables importantes y su uso, restricciones y limitaciones y de otra información importante
4. Una historia del desarrollo que incluya:
  - a. el diseñador del módulo (autor)

b. el revisor (auditor) y la fecha

c. fechas de modificación

Los comentarios descriptivos se encuentran en el cuerpo del código fuente y se usan para describir las funciones de procesamiento.

### 2.1.3. Declaración de datos

Debemos tomar una decisión sobre el orden en la declaración de los datos. Éste hace que los datos sean fáciles de descubrir, comprobar y mantener. Cuando se declaran múltiples nombres de variables en una sentencia, merece la pena ponerlos en orden alfabético. Si el diseño describe una estructura de datos compleja, se deben usar comentarios para explicar las particularidades inherentes a la implementación en el lenguaje de programación.

### 2.1.4. Construcción de sentencias

La construcción del flujo lógico del software se establece durante el diseño. La construcción de sentencias individuales es parte de la codificación. La construcción de sentencias se debe basar en una regla general: cada sentencia debe ser simple y directa, el código no debe ser retorcido.

Muchos lenguajes de programación permiten disponer múltiples sentencias en una misma línea. El ahorro de espacio que esto implica no siempre está justificado por la pobre legibilidad del resultado.

Las sentencias de código fuente se pueden simplificar al:

1. Evitar el uso de complicadas comparaciones condicionales
2. Eliminar las comparaciones con condiciones negativas
3. Evitar un gran anidamiento de bucles o de condiciones
4. Usar paréntesis para clarificar las expresiones lógicas o aritméticas
5. Usar espacios y/o símbolos claros para aumentar la legibilidad del contenido de la sentencia
6. Pensar "¿Podría yo entender esto si no fuera la persona que lo codificó?"

### 2.1.5. Entrada/salida

La forma en que se implementa la E/S puede ser una característica determinante de la aceptación del sistema por una comunidad de usuarios. El estilo de la entrada salida variará con el grado de integración humana. Se deben considerar una serie de principios para el diseño y la codificación de la E/S:

1. Validar todos los datos de entrada
2. Comprobar las importantes combinaciones de elementos de entrada

3. Mantener el formato de entrada simple
4. Etiquetar las peticiones interactivas de entrada, especificando las opciones posibles o los valores límite.
5. Etiquetar todas las salidas y diseñar todos los informes.

 *Para saber más...*

*En resumen,*

¿A quién interesa el código fuente?

- Autores del propio código
- Otros desarrolladores del proyecto
- Clientes de la API del proyecto

¿Por qué documentarlo?

- Mantenimiento
- Reutilización

¿Qué documentar?

- Obligatorio:
  - o Clases y paquetes
  - o Constructores, métodos, atributos...
- Conveniente:
  - o Fragmentos no evidentes
  - o Bucles, algoritmos...



### 3. UNA HERRAMIENTA: JAVADOC

Javadoc es una herramienta para el lenguaje java, que permite generar la documentación del API (Application Programming Interface) de los programas desarrollados.

Esta herramienta analiza (parsea) las declaraciones y los comentarios de la documentación de un conjunto de ficheros fuente y produce páginas html como documentación externa, describiendo clases, clases internas (subclases), interfaces, constructores y atributos de la clase (también llamados fields o campos).

Javadoc permite tener documentación interna dentro del código, y documentación externa como documentación técnica en el que se detalla la aplicación java.

#### 3.1. Cómo hacer comentarios javadoc

Los comentarios en Java pueden ser de una línea, de varias o Javadoc. Los Javadoc son como los multilínea pero comienzan con 2 asteriscos: `/**` Una línea

```
/*
 * Comentario Multilínea
 */

/**
 * Comentario Javadoc
 */
```

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 ítem a comentar. En caso contrario la herramienta de generación automática no lo reconocerá.

Los comentarios javadoc tienen dos partes:

- La parte de la descripción
- La parte de etiquetas o *tags*

Ejemplo:

```
/**
 *
 * Descripción principal ( Texto / HTML )
 *
 * Tags ( Texto / HTML )
 */
```

```

/**
 * Returns the index of the first occurrence of the specified element in
 * this vector, searching forwards from <code>index</code>, or returns -1 if
 * the element is not found.
 *
 * @param o element to search for
 * @param index index to start searching from
 * @return the index of the first occurrence of the element in
 *         this vector at position <code>index</code> or later in the vector;
 *         <code>-1</code> if the element is not found
 * @throws IndexOutOfBoundsException if the specified index is negative
 * @see    Object#equals(Object)
 */
public int indexOf(Object o, int index) ...

```

### indexOf

```
public int indexOf(Object o,
                 int index)
```

Returns the index of the first occurrence of the specified element in this vector, searching forwards from `index`, or returns -1 if the element is not found.

**Parameters:**

`o` - element to search for  
`index` - index to start searching from

**Returns:**

the index of the first occurrence of the element in this vector at position `index` or later in the vector; -1 if the element is not found.

**Throws:**

[IndexOutOfBoundsException](#) - if the specified index is negative

**See Also:**

[Object.equals\(Object\)](#)

### 3.2. Uso de etiquetas javadoc

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

Sintaxis	Descripción
<b>@author nombre</b>	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.

<b>@deprecated comentario</b>	<p>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, esto es sólo una advertencia que damos a los usuarios de nuestras API's para prevenirles de que en un futuro podrán surgir incompatibilidades si seguen usándolas.</p> <p>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 qué versión de nuestra API está desautorizada y por quién se debería sustituir. A partir de Java 1.2 podemos usar <i>@link</i> para referenciar por quién debemos hacer la sustitución.</p> <p>Esta etiqueta actúa exactamente igual que <i>@throws</i>.</p>
<b>@link nombre etiqueta</b>	<p>Inserta un enlace autocontenido que apunta a nombre. Esta etiqueta acepta exactamente la misma sintaxis que la etiqueta <i>@see</i>, 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 caracteres para separarla del resto del texto in-line, si necesitas emplear el caracter “}” dentro de la etiqueta debes usar la notación HTML <code>&amp;#125;</code>.</p> <p>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.</p> <p>Veamos un ejemplo de cómo crear dentro de nuestra documentación un enlace in-line al método <i>getComponentAt(int, int)</i>:</p> <pre>@deprecated Usar el método {@link #getComponentAt(int, int) getComponentAt}</pre> <p>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):</p> <pre>Usar el método &lt;ahref="Component.html#getComponentAt(int,int)"&gt;getComponentAt&lt;/a&gt;</pre> <p>el cual aparecerá en la página HTML como:</p> <pre>Usar el método getComponentAt</pre>

<p><b>@param</b> parámetro descripción</p>	<p>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.</p> <p>Se aplica a parámetros de constructores y métodos. Describe los parámetros del constructor/método.</p> <p>Ejemplo:</p> <pre> /**  * Borra de una lista de items todos sus elementos  *  * @param formIndex indice del primer elemento a borrar  * @param toIndex indice del ultimo elemento a borrar  */ protected void removeRange ( int formIndex , int toIndex ){     ... }</pre>
<p><b>@return</b> descripción</p>	<p>Añade a la sección “Returns” de la documentación HTML que va a generar la descripción del tipo que devuelve el método. Se aplica a métodos. Describe el valor de retorno del método. Se incluye la descripción del valor de retorno.</p> <p>Ejemplo:</p> <pre> /*  *  * ....  * @param s1 Texto que ocupa la  * @param s2 Texto que ocupa el  * @param s3 Texto que ocupa la  * @return La cadena conformada  */ public String concatena ( String s1 , String s2 , String s3) {     ... }</pre> <p>Ejemplo:</p> <pre> /*  * Comprueba si el vector no tiene componentes  * @return &lt;code&gt;true &lt;code&gt; si no tiene componentes  * &lt;code&gt;false &lt;/code&gt; en otro caso  */ public booleana concatena () {     return elementCount ==0; }</pre>

<b>@see referencia</b>	<p>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.</p> <p>Esta etiqueta se puede conformar de tres maneras diferentes, siendo la última forma la más utilizada.</p>
	<p><b>@see "string"</b></p> <p>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 (").</p> <p>Por ejemplo:</p> <pre>@see "The Java Programming Language"</pre> <p>que genera el siguiente texto HTML:</p> <pre>See Also: "The Java Programming Language"</pre>
	<p><b>@see &lt;a href="URL#value"&gt;label&lt;/a&gt;</b></p> <p>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 "&lt;".</p> <p>Por ejemplo:</p> <pre>@see &lt;a href="spec.html#section"&gt;Java Spec&lt;/a&gt;</pre> <p>que genera el siguiente enlace:</p> <pre>See Also: Java Spec</pre>

	<p><b>@see package.class#member texto</b></p> <p>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, esto es, el nombre de, por ejemplo, la pareja paquete#método.</p> <p>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. Veamos a continuación varios ejemplos usando la etiqueta @see.</p> <p><i>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.</i></p> <pre>@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 &lt;a href = "spec . html"&gt; Java Spec &lt;/a&gt; // Java Spec @see "The Java Programming Language"//Language "The Java Programming"</pre>
<b>@since texto</b>	<p>Indica con texto desde cuando 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:</p> <pre>@since JDK1 .1</pre>
<b>@serial field-description</b>	<p>Su uso estáa 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, se usa sólo cuando se tenga una variable serializable.</p>
<b>@serialField field- name field- typefield- description</b>	<p>Documenta un componente <i>ObjectStreamField</i> de un miembro serialPersistentFields de una clase Serializable. Esta etiqueta se debería usar para cada componente <i>ObjectStreamField</i>.</p>
<b>@serialData data- description</b>	<p>Se emplea para describir los datos escritos por el método <i>writeObject</i> y todos los datos escritos por el método <i>Externalizable.writeExternal</i>. Esta etiqueta puede ser usada en aquellas clases o métodos que intervengan los métodos <i>writeObject</i>, <i>readObject</i>, <i>writeExternal</i>, y <i>readExternal</i>.</p>

<b>@throws</b> nombre- clase descripción	<p>Como ya se apuntó, esta etiqueta es la gemela de <b>@exception</b>. 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. Se aplica a constructores y métodos. Describe posibles excepciones. Un <b>@throws</b> por cada posible excepción.</p> <p>Ejemplo:</p> <pre> /**  * Analiza el argumento string como un número decimal  * &lt;code&gt;long &lt;/code&gt;  *  * @params un &lt;code&gt; String &lt;/code&gt; contiene el &lt;code&gt;long &lt;/code&gt;  *  * @return el &lt;code&gt;long &lt;/code&gt; modificado  *  * @exception NumberFormatException  * si el string no contiene un &lt;code&gt;long &lt;/code&gt; analizable.  */ public static long parseLong (String s) throws NumberFormatException { ... } </pre>
<b>@version</b> version	<p>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.</p>

Según dónde vayamos a poner las etiquetas, se pueden usar unas u otras:

Qué se comenta	Tipo de tag
<b>Página Overview</b>	<p>Las etiquetas que se pueden emplear en esta sección son:</p> <p><b>@see</b>, <b>@link</b>, <b>@since</b>.</p> <p>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 <i>overview.html</i> es la que nos da una idea global de las clases y paquetes que conforman nuestra API (ver especificaciones html de Sun).</p>

<b>Paquetes</b>	<p>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 <i>package.html</i>.</p> <p>Las etiquetas que podemos utilizar en este caso son:</p> <p>@see, @link, @since, @deprecated</p>
<b>Clases, interfaces</b>	<p>Disponemos de las siguientes etiquetas:</p> <p>@see, @link, @since, @deprecated, @author, @version</p> <p>Ejemplo:</p> <pre> /**  * A class representing a window on the screen .  * For example :  * &lt;pre&gt;  * Window win = new Window (parent);  * win . show ();  * &lt;/pre&gt;  *  * @author Sami Shaio  * @version %I %, %G %  * @see java . awt . BaseWindow  * @see java . awt . Button  */ class Window extends BaseWindow {     ... } </pre>
<b>variables</b>	<p>Las etiquetas que podemos utilizar en esta ocasión son:</p> <p>@see, @link, @since, @deprecated, @serial, @serialField</p> <p>Ejemplo:</p> <pre> /**  * The X- coordinate of the component .  * @see # getLocation ()  */ int x = 1263732; </pre>



**Métodos,  
constructores**

En este caso disponemos de:

`@see`, `@link`, `@since`, `@deprecated`, `@param`, `@return`, `@throws` (`@exception`), `@serialData`

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 StringIndexOutOfBoundsException 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 ) {
    ...
}
```

**3.3. Ejemplo de etiquetado**

A continuación se muestra un ejemplo de documentación de una clase usando etiquetas:

```
/**
 * <h2> Clase Empleado, se utiliza para crear y leer empleados de una BD </h2>
 *
 * Busca información de javadoc en <a href=http://google.com>GOOGLE</a>
 * @see <a href=http://www.google.com>Google</a>
 * @version 1-2014
 * @author ARM * @since 1-1-2014
 */
public class Empleado{
    /**
     * Atributo Nombre del empleado
     */
    private String nombre;
    /**
     * Atributo Apellido del empleado
     */
    private String apellido;
    /**
```

```
* Atributo salario del empleado
*/
private double salario;
/**
 * Constructor con 3 parámetros
 * Crea objetos empleado, con nombre, apellidos y salario
 * @param nombre Nombre del empleado
 * @param apellido Apellido del empleado
 * @param salario Salario del empleado
 public Empleado(String nombre, String apellido, double salario){
     this.nombre=nombre;
     this.apellido=apellido;
     this.salario=salario;
 }
 //Métodos públicos
 /**
 * Sube el salario al empleado
 * @see Empleado
 * @param subida
 */
 public void subidasalario(double subida){
     salario = salario + subida;
 }
 //Métodos privados
 /**
 * Comprueba que el nombre no esté vacío
 * @return <ul>
 *         <li> true: el nombre es una cadena vacía </li>
 *         <li> false: el nombre no es una cadena vacía </li>
 *     </ul>
 */
 private boolean comprobar(){
     if (nombre.equals("")){
         return false;
     }
     Return true;
 }
 }
```

(fuente: Garceta, 2014:186-187)

Recuerda que los comentarios de documentación Javadoc deben ubicarse ANTES de las declaraciones de las clases, de los métodos o de los atributos.

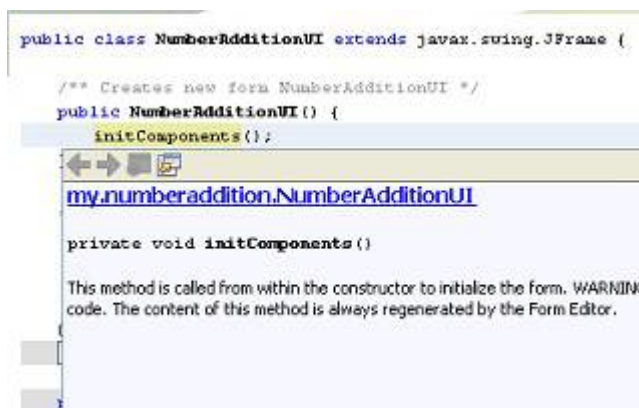
### 3.4. Javadoc en Netbeans

Netbeans permite crear la documentación interna, de forma sencilla, para ello veremos como añadir comentarios y la documentación html con javadoc.

#### 3.4.1. Cómo visualizar el API de las librerías estándar

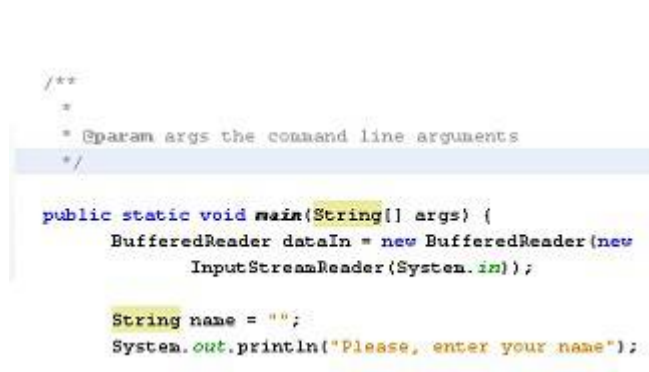
Seleccionar el elemento. y Pulsar CTRL + SHIFT + Espacio o Menú Fuente - Mostrar Documentación.

El Javadoc para este elemento es mostrado mediante una ventana emergente.



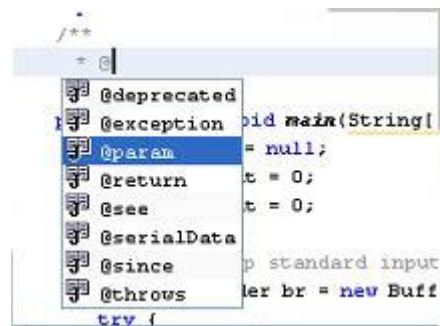
#### 3.4.2. Cómo comentar un método

Podemos hacer salgan los argumentos para comentarlos nos podremos delante del método y escribiremos /\*\* y pulsaremos INTRO.



#### 3.4.3. Cómo añadir tags a un método

Pondremos el símbolo arroba y nos mostrará los tags disponibles.



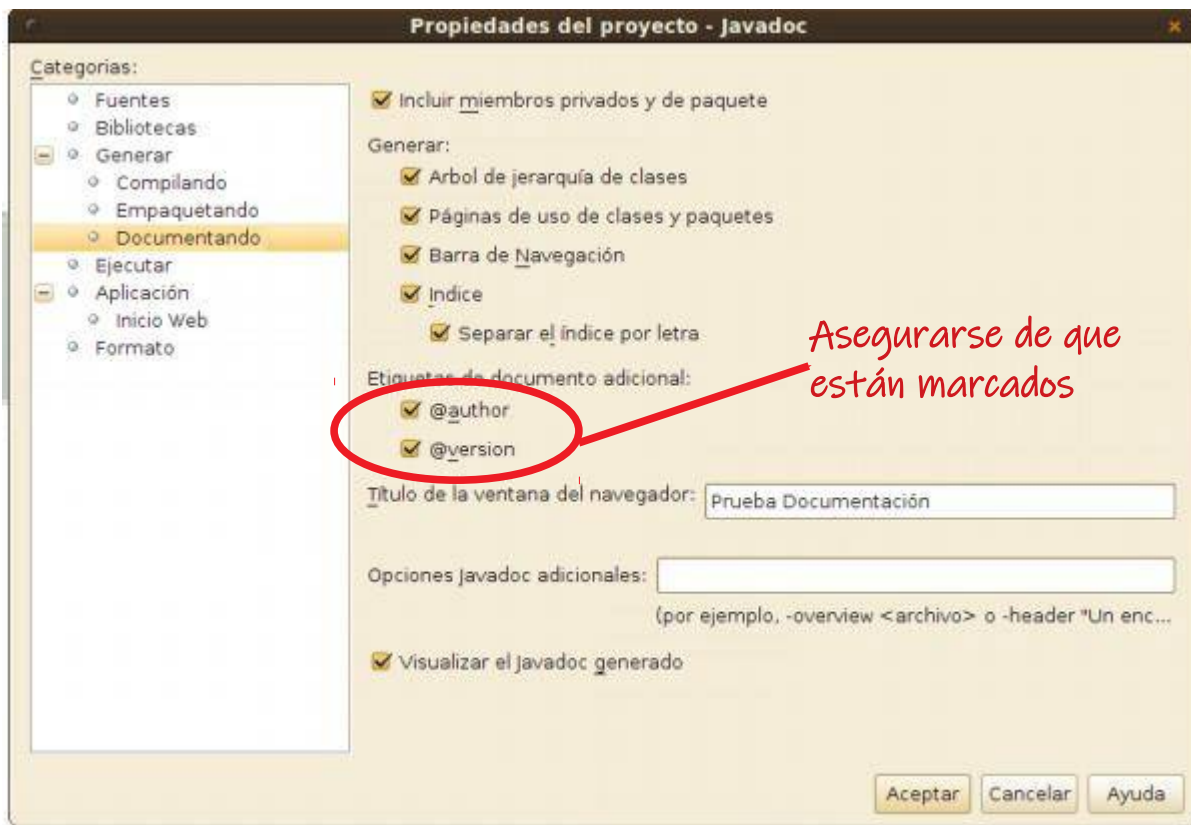
### 3.4.4. Cómo generar un javadoc

Podremos generar una página web con la documentación, a través del Menú Ejecutar - Generar Javadoc.



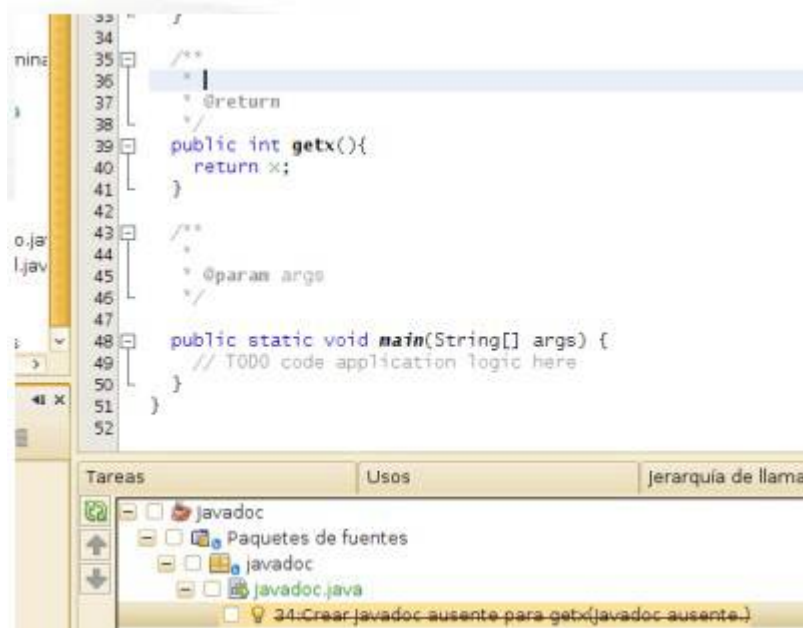
### 3.4.5. Cómo personalizar el formato de javadoc

Para personalizar el formato de javadoc, seleccionamos el nombre del proyecto, y vamos al Menú Archivo - Proyecto Propiedades (Javadoc). Entre otras cosas permite poner el título a la ventana que genera Javadoc:



### 3.4.6. Cómo buscar métodos sin documentación

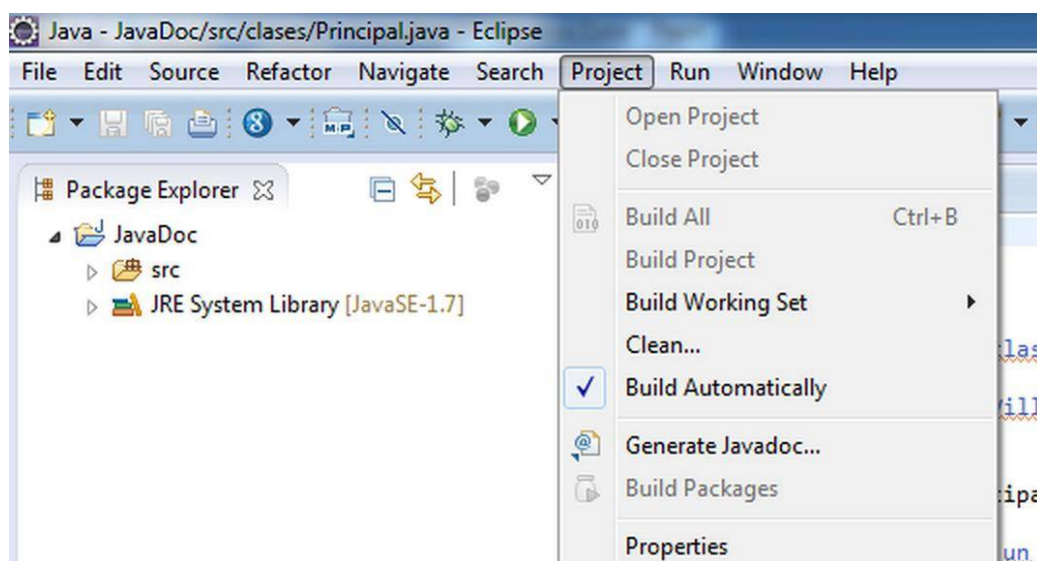
Ir al Menú Herramientas - Analizar Javadoc. Esto permitirá ver que métodos no están comentados. Y marcando reparar nos genera los comentarios. Seguidamente lo tacha para indicarnos que ya tiene sus comentarios.



### 3.5. Javadoc en Eclipse

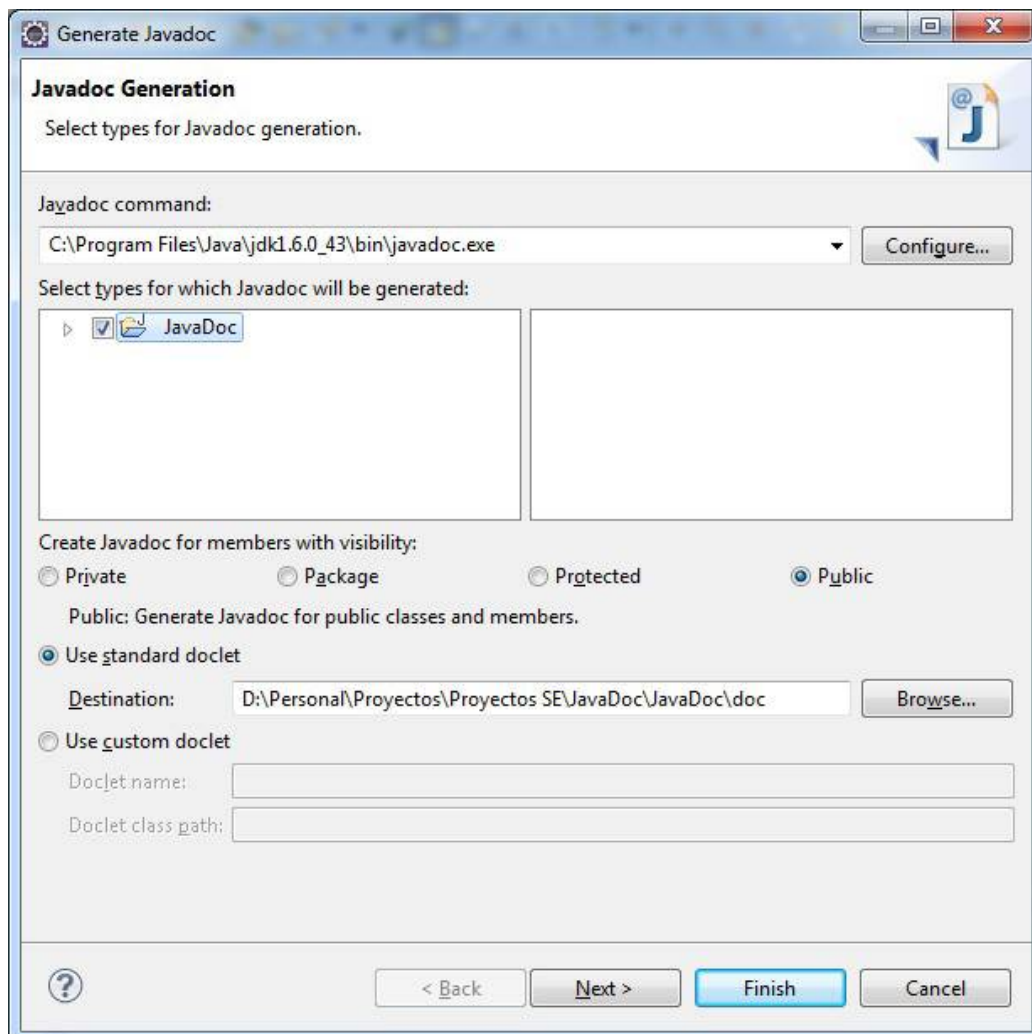
Al igual que en Netbeans, Javadoc puede ser usado en Eclipse. De hecho, la mayor parte de los entornos tienen un enlace para configurarlo y usarlo.

Para ejecutar Javadoc desde Eclipse abre el menú “Project” > “Generate Javadoc”.



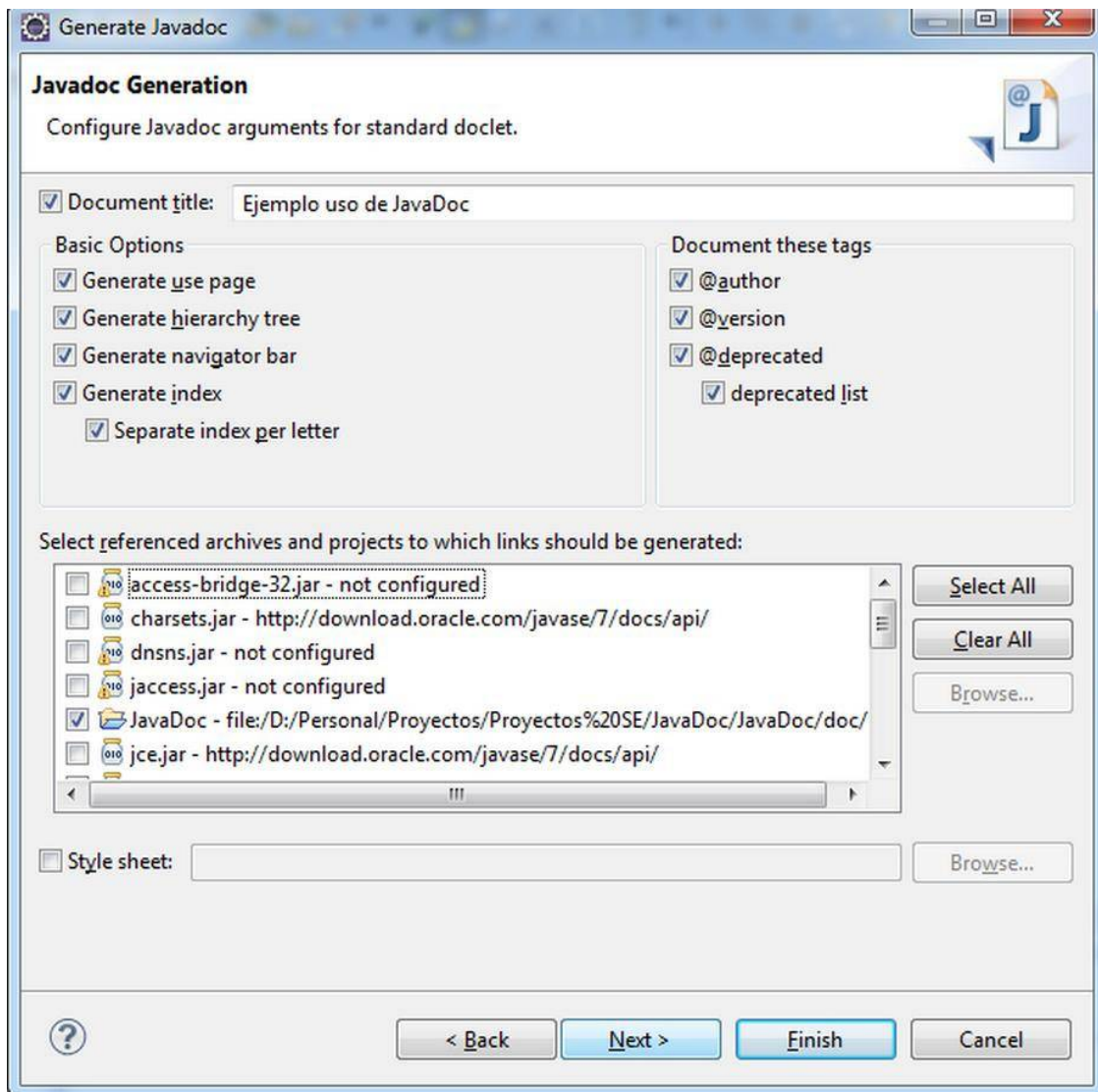
En la ventana que se muestra pedirá la siguiente configuración:

- En “Javadoc Command” debemos indicar dónde está el archivo ejecutable de Javadoc (javadoc.exe). Pulsa el botón “Configure” para buscarlo dentro de la carpeta donde se encuentra instalado el JDK, y dentro de esa carpeta elige la carpeta bin.
- En los dos cuadros inferiores se elige el proyecto y las clases a documentar
- Selecciona la visibilidad de los elementos a documentar. Con “Private” se documentarán todos los miembros públicos, privados y protegidos
- Finalmente se indica la carpeta destino donde guardar el código HTML. Ten en cuenta que es conveniente poner un nombre distinto al que viene por defecto porque con *doc* hay veces que no se cargan los estilos.

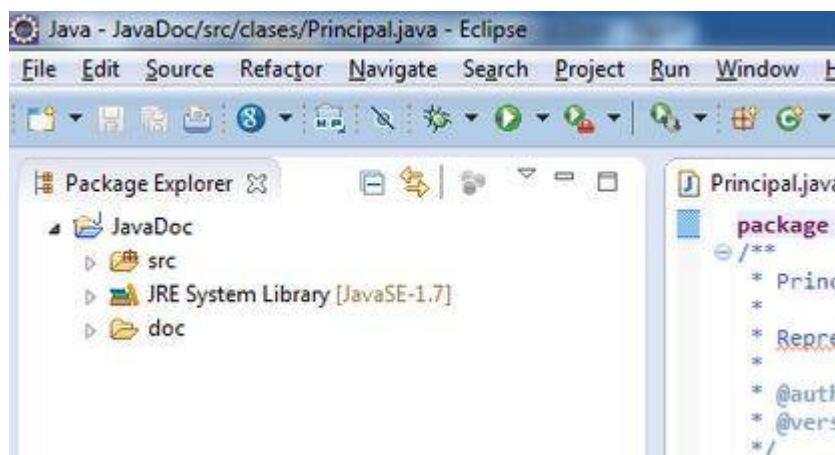


Ahora pulsa “Next” y en la siguiente ventana indica el título del documento html que generará, además permite elegir opciones para generar páginas html. Como mínimo selecciona la barra de navegación y el índice.

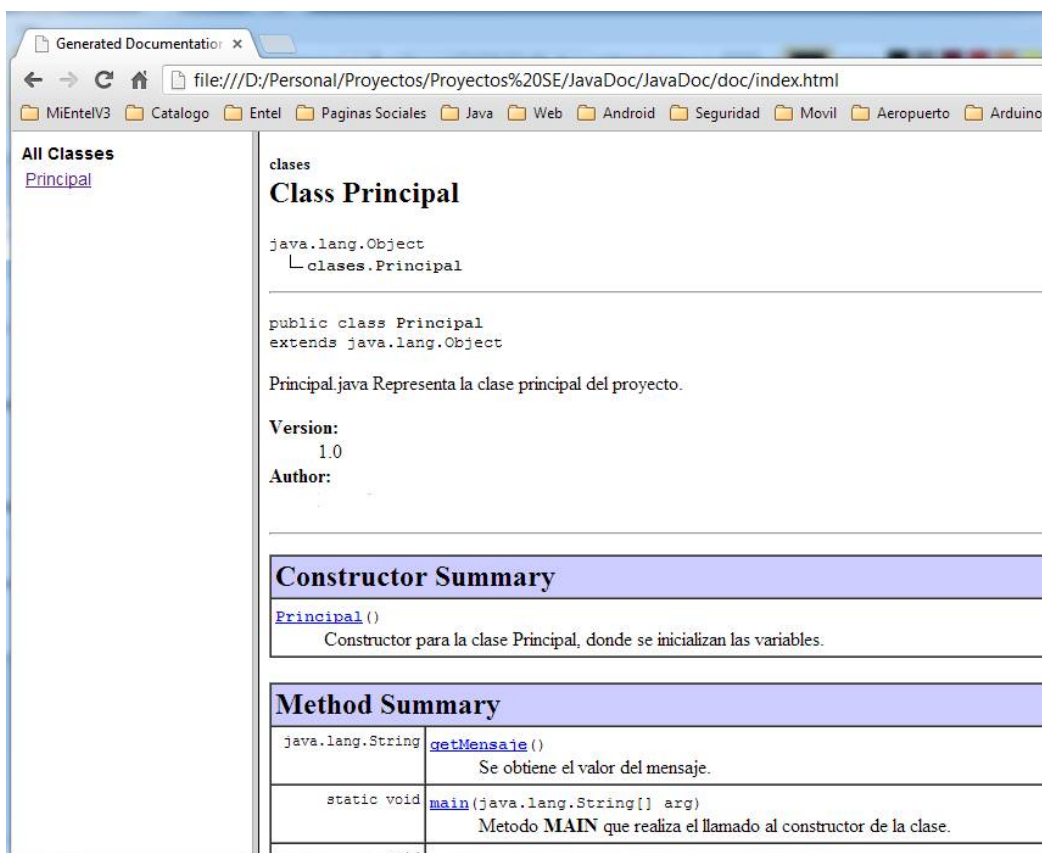




Verás que en tu proyecto aparece ahora una carpeta “doc”. Dentro de ella se encuentran los archivos generados, son archivos html, junto con los estilos y demás que vienen por defecto para la documentación en java.



Si vamos al navegador de archivos y entramos en la carpeta doc, al abrir el archivo *index.html* veremos la siguiente pantalla con toda la documentación generada, además podremos navegar fácilmente entre las clases, métodos y demás.



#### 4. BIBLIOGRAFÍA Y ENLACES

- Aldarias, F. (2012): “Entornos de desarrollo”, CEEDCV
- Amescua, A.; Cuadrado, JJ; Ernica, E. (2003): *Análisis y Diseño Estructurado y Orientado a Objetos de Sistemas Informáticos*, Mcgraw-hill
- Casado, C. (2012): *Entornos de desarrollo*, RA-MA, Madrid
- Ramos, A.; Ramos, M.J. (2014): *Entornos de desarrollo*, Garceta, Madrid