

## **EL API JAVA**

El término "api java" tiene varias acepciones. La que usamos de forma inmediata, aunque no del todo correcta, es la que significa "documentación de ayuda de las clases del sistema".

Se accede rápidamente al API java haciendo búsquedas directamente desde google, por ejemplo, para ver la documentación de la clase Scanner podemos teclear en google palabras del estilo "api java Scanner"

## **JAVADOC**

Javadoc es una utilidad que nos permite documentar nuestras clases siguiendo el mismo formato estándar que la documentación del api java. Recuerda que decíamos que el JDK es un conjunto de utilidades para el programador Java. Por el momento utilizamos Javac, Java y JShell. Ahora sumamos a la lista también JavaDoc.

Echa un vistazo rápido a este documento, con que pruebes el primer ejemplo que genera ficheros de documentación es suficiente por el momento.

### **Comentarios java**

Desde el punto de vista de "quien va a leer" los comentarios distinguimos dos tipos:

- Comentarios Internos: Para explicar el propósito de sentencias o grupos de sentencias. Estos comentarios son útiles para el propio autor del código, y para otros que quieran leer y modificar ese código. Los comentarios pueden ser:
  - o de una línea con `//`
  - o de varias líneas con `/* */`

Ejemplo:

```
class Unidad2 {  
  
    public static void main() {  
        // esto es un comentario de una línea  
        //otro comentario de una línea  
  
        /*comentario multilínea  
        aquí sigue el comentario  
        y sigue  
        esta es la última línea del comentario multilínea*/  
        System.out.println("Hola mundo");  
    }  
}
```

- Comentarios Externos(comentarios javadoc): Comentarios explicando qué hace una "pieza" cerrada de código. Estos comentarios son útiles para quien quiere utilizar esta "pieza" en su propio programa, y que por tanto está necesita saber qué hace, no cómo se las ha arreglado el programador para conseguir ese resultado. Nos referimos a los comentarios javadoc. Por tanto el objetivo de los comentarios Javadoc es documentar el código de una clase como una caja negra, no interesa "cómo" lo hace el código, si no "qué hace" y cómo puedo usar dicha clase. Esta documentación se genera en un formato standard coincidente con el formato que utiliza la API Java.

**UNA CONCLUSIÓN:** No utilices comentarios javadoc para explicar cómo funciona el código.

**Ejercicio:** consulta la documentación de la api Java relativa a la clase String y la clase Math y observa que no son más que páginas html con idéntica estructura.

Utilizando la utilidad javadoc, la documentación de nuestras clases tendrá también la estructura anterior.

**Ejercicio:** Consulta el código fuente de por ejemplo la clase Scanner para ver sus javadoc. Hay muchas formas de ver el código fuente de una clase. Lo más fácil es buscar en google "java scanner código fuente" y ver el código de la clase del openjdk

### Comentarios Javadoc.

Constan de tres elementos:

- Los símbolos delimitadores de un comentario `/** */`
- Los tags, etiquetas especiales precedidas por `@`
- Texto explicativo

**Ejemplo:** clase con Javadoc simple, sin utilizar tags.

Se puede comentar cualquier clase, atributo o método simplemente incluyendo un comentario Javadoc justo antes de dicha clase, atributo o método.

```
/**
 * Representa un empleado de un departamento de la empresa
 * no se permite que un empleado pertenezca a varios departamentos
 */
public class Empleado{

    /**
     * el nombre completo del Empleado.
     */
    private String name;

    /**
     * el nombre del departamento en que trabaja el empleado.
     */
    public String departamento;

    /**
     * Crea un nuevo empleado a partir de su nombre.
     * El nombre debería incluir nombre de pila y apellidos.
     */
    public Empleado(String name) {
        this.name = name;
    }
}
```

**Ejemplo:** generación de la documentación de la clase anterior.



Suponiendo que tenemos la clase del ejemplo anterior en un fichero Empleado.java, simplemente, pasamos este fichero como parámetro de la herramienta javadoc

```

C:\borrame>javadoc Empleado.java
Loading source file Empleado.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0_20
Building tree for all the packages and classes...
Generating .\Empleado.html...
Empleado.java:22: warning: no @param for name
    public Empleado(String name) {
           ^
Generating .\package-frame.html...
Generating .\package-summary.html...
Generating .\package-tree.html...
Generating .\constant-values.html...
Building index for all the packages and classes...
Generating .\overview-tree.html...
Generating .\index-all.html...
Generating .\deprecated-list.html...
Building index for all classes...
Generating .\allclasses-frame.html...
Generating .\allclasses-noframe.html...
Generating .\index.html...
Generating .\help-doc.html...
1 warning
C:\borrame>

```

si hago 2 clic sobre Empleado.html

 deprecated-list	23/12/2013 10:30	Chrome HTML Do...	4 KB
 Empleado	23/12/2013 16:38	Chrome HTML Do...	10 KB

Observo la documentación generada

PACKAGE
CLASS
TREE
DEPRECATED
INDEX
HELP

PREV CLASS
NEXT CLASS
FRAMES
NO FRAMES
ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

### Class Empleado

java.lang.Object  
Empleado

```
public class Empleado
extends java.lang.Object
```

Representa un empleado de un departamento de la empresa no se permite que un empleado pertenezca a varios departamentos

Field Summary

Fields

Modifier and Type	Field and Description
java.lang.String	departamento el nombre del departamento en que trabaja el empleado.

Constructor Summary

Constructors

Constructor and Description
Empleado(java.lang.String name) Crea un nuevo empleado a partir de su nombre.

Method Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

departamento

public java.lang.String departamento  
el nombre del departamento en que trabaja el empleado.

Constructor Detail

Empleado

public Empleado(java.lang.String name)  
Crea un nuevo empleado a partir de su nombre. El nombre debería incluir nombre de pila y apellidos.

Fíjate como en Field Summary sólo aparecen los atributos “public”, ya que queremos una perspectiva de “caja negra” de la clase, es lógico que un detalle “interno” como un atributo private no sea visto desde el exterior.

Ejemplo: genera el javadoc del siguiente código al que deliberadamente se le retiró a la clase el calificativo de public. Observarás que no es posible generar documentación. Nos dice que no encuentra clase public o protected. Para nosotros por el momento protected es equivalente a public.

```

/**
 * Representa un empleado de un departamento de la empresa
 * no se permite que un empleado pertenezca a varios departamentos

```

```

*/
class Empleado{

    /**
     * el nombre completo del Empleado.
     */
    private String name;

    /**
     * el nombre del departamento en que trabaja el empleado.
     */
    public String departamento;

    /**
     * Crea un nuevo empleado a partir de su nombre.
     * El nombre debería incluir nombre de pila y apellidos.
     */
    public Empleado(String name) {
        this.name = name;
    }
}

```

Ejemplo: genera el javadoc del siguiente código. Ahora la clase es public, pero los atributos y el constructor es private. Observarás que lo que no es public no tiene documentación asociada.

```

/**
 * Representa un empleado de un departamento de la empresa
 * no se permite que un empleado pertenezca a varios departamentos
 */
public class Empleado{

    /**
     * el nombre completo del Empleado.
     */
    String name;

    /**
     * el nombre del departamento en que trabaja el empleado.
     */
    String departamento;

    /**
     * Crea un nuevo empleado a partir de su nombre.
     * El nombre debería incluir nombre de pila y apellidos.
     */
    Empleado(String name) {
        this.name = name;
    }
}

```

## Comentarios Javadoc que incluyen tags

El uso de tags lo vemos como una simple curiosidad, aquí vemos un ejemplo sin más trascendencia que utiliza las etiquetas @param y @return

@param nombreParámetro descripción.

Para describir un parámetro de un método.

@return descripción.

Describe el valor de salida de un método.

Ejemplo:

```

/**
 * Representa un empleado de un departamento de la empresa
 * no se permite que un empleado pertenezca a varios departamentos
 */
public class Empleado{

    /**
     * el nombre completo del Empleado. Este comentario es inútil por ser nombre private
     */
    private String nombre;
    private int sueldo;

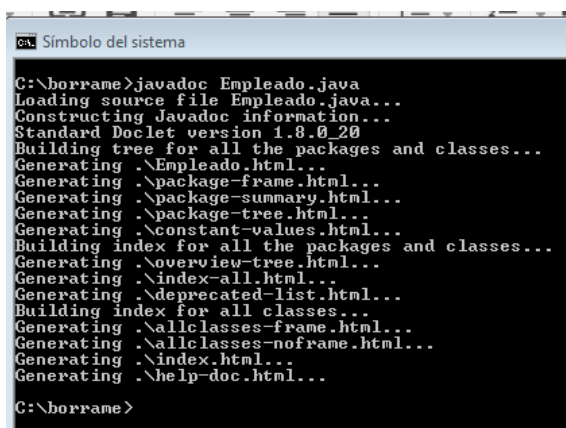
    /**
     * el nombre del departamento en que trabaja el empleado.
     */
    public String departamento;

    /**
     * Crea un nuevo empleado a partir de su nombre.
     * @param nombre nombre completo del empleado.
     */
    public Empleado(String nombre) {
        this.nombre = nombre;
    }

    /**
     * Devuelve el sueldo de un empleado.
     * @return el sueldo como un valor int.
     */
    public int getSuelo(){
        return sueldo;
    }
}

```

Si ahora vuelvo a generar para el nuevo fichero la nueva documentación



```

C:\borrame>javadoc Empleado.java
Loading source file Empleado.java...
Constructing Javadoc information...
Standard Doclet version 1.8.0_20
Building tree for all the packages and classes...
Generating \Empleado.html...
Generating \package-frame.html...
Generating \package-summary.html...
Generating \package-tree.html...
Generating \constant-values.html...
Building index for all the packages and classes...
Generating \overview-tree.html...
Generating \index-all.html...
Generating \deprecated-list.html...
Building index for all classes...
Generating \allclasses-frame.html...
Generating \allclasses-noframe.html...
Generating \index.html...
Generating \help-doc.html...
C:\borrame>

```

Y hago 2 clic en Empleado.html, observo novedades, por ejemplo, la documentación del parámetro del constructor y del retorno del método getSuelo():

## Constructor Detail

### Empleado

```
public Empleado(java.lang.String nombre)
```

Crea un nuevo empleado a partir de su nombre.

#### Parameters:

nombre - nombre completo del empleado.

## Method Detail

### getSueldo

```
public int getSueldo()
```

Devuelve el sueldo de un empleado.

#### Returns:

el sueldo como un valor int.

## Comentarios Javadoc para campos private

Aunque normalmente lo que se quiere es comentar con estilo javadoc todo lo public, también es posible comentar lo private y generar su documentación html. Simplemente usamos el modificador -private al invocar a javadoc

```
E:\Programacion\borrar>javadoc Empleado.java -private
```

```
/**
 * Representa un empleado de un departamento de la empresa
 * no se permite que un empleado pertenezca a varios departamentos
 */
public class Empleado{

    /**
     * el nombre completo del Empleado.Con -private se ven nombre y sueldo
     */
    private String nombre;
    private int sueldo;

    /**
     * el nombre del departamento en que trabaja el empleado.
     */
    public String departamento;

    /**
     * Crea un nuevo empleado a partir de su nombre.
     * @param nombre nombre completo del empleado.
     */
    public Empleado(String nombre) {
        this.nombre = nombre;
    }
}
```

```

    }

    /**
     * Devuelve el sueldo de un empleado.
     * @return el sueldo como un valor int.
     */
    public int getSueldo(){
        return sueldo;
    }
}

```

y nos fijamos en las novedades

Fields	
Modifier and Type	Field and Description
java.lang.String	<b>departamento</b> el nombre del departamento en que trabaja el empleado.
private java.lang.String	<b>nombre</b> el nombre completo del Empleado. Con -private se ven nombre y sueldo
private int	<b>sueldo</b>