

## Análisis de los pasos de Javadoc partiendo desde el archivo fuente hasta la generación de los archivos html

El uso de la herramienta Javadoc dentro de la línea de comandos tiene la siguiente sintaxis.

**javadoc** [ opciones ] [ paquetes ] [ archivos fuente ] [ subpaquetes ] [ argFiles ]

La explicación de cada uno de los parámetros es la siguiente:

### Opciones

El proceso de Javadoc ocupa un conjunto de herramientas opcionales llamados *doclets*. Básicamente estas herramientas determinan el producto final en html y proveen de una forma de ser sustituidas por el usuario.

Por otra parte, cualquier *doclet*, ya sea el default o alguno especificado por el usuario, debe de cumplir con la compatibilidad con las siguientes opciones o modificadores. La explicación de cada uno de ellos se encuentra en el anexo 1.1. Todas las opciones son *case sensitive* al igual que sus argumentos.

-1.1	-header	-private
-author	-help	-protected
-bootclasspath	-helpfile	-public
-bottom	-J	-quiet
-breakiterator	-link	-serialwarn
-charset	-linkoffline	-source
-classpath	-linksource	-sourcepath
-d	-locale	-splitindex
-docencoding	-nocomment	-stylesheetfile
-docfilessubdirs	-nodeprecated	-subpackages
-doclet	-nodeprecatedlist	-tag
-docletpath	-nohelp	-taglet
-doctitle	-noindex	-tagletpath
-encoding	-nonavbar	-title
-exclude	-noqualifier	-use
-excludedocfilessubdir	-nosince	-verbose
-extdirs	-notree	-version
-footer	-overview	-windowtitle
-group	-package	

### Paquetes

Se puede especificar una serie de paquetes, separados por una línea. En este caso, los *wildcards* no son permitidos (ejemplo \*.java) por lo que si se necesita recursividad, es decir, procesar tanto el paquete padre como todos sus sucesores, es necesario usar la opción *-subpackages*.

### Archivos Fuente

Se especifican uno o mas archivos fuentes, separados por espacios. Es permitido el uso de los *wildcards* para agrupar distintos archivos de una sola vez. Es importante señalar que Javadoc únicamente procesará aquellos archivos con nombre de clase válido, siguiendo los estándares de Java. De esta manera, los nombres no deben de comenzar con números, contener guiones (-) o algún otro carácter ilegal. Esto puede servir como ventaja, pues se pueden excluir ciertos archivos a los cuales no se quiere procesar, tan solo con llamarle de alguna manera que no los reconozca.

La dirección de los archivos puede ser relativa, absoluta o calificada. Dentro del *path* pueden también existir *wildcards*.

### Subpaquetes

Si se especifica la opción *-subpackages* indica a Javadoc que también documente de manera recursiva los paquetes sucesores del paquete que le pasamos como parámetro.

### ArgFiles

Para mayor facilidad, se puede crear un archivo con todas las opciones, parámetros, paquetes y nombres de archivo, para después pasarlo como argumento dentro de la opción *@nombreDelArchivo*.

## Procesamiento de los archivos fuente

Javadoc procesa todos los archivos especificados que terminen en *.java*, mas algún otro especificado de manera especial. También soporta el uso de los *wildcards* para una mayor facilidad de documentar grupos de archivos. Antes de procesar un archivo, se ejecutan las siguientes comprobaciones:

1. El nombre del archivo, al quitarle la extensión *.java*, deben de corresponder con las reglas de nombre de clases, es decir, no deben de comenzar con números, contener guiones (-) o algún otro carácter ilegal.
2. El *path* relativo especificado es nombre legal de paquete, después de convertir sus separadores a puntos.
3. El enunciado en donde se declara a que paquete pertenece corresponde con las reglas de declaración de paquetes.

En el conjunto de archivos HTML que resultan, se tienen referencias a todas las clases, para establecer las distintas relaciones entre ellas (paquetes, herencia, composición, etc).

También existen referencias hacia adentro de la clase, para documentar y hacer más fácil la navegación de los métodos, atributos y paquetes.

En resumen, las referencias existen en los siguientes casos:

- En declaraciones (Retornos, parámetros e instancias).
- En referencias tipo "See Also" (generadas por el *tag @see*)
- En texto mediante especificación directa del usuario (con el *tag @link*)
- Nombres de Exception generados por el *tag @throws*.
- Tablas de índice de las clases, paquetes y miembros.
- Árboles de herencia y de paquetes.
- El índice (*index.html*)

## Detalles del Proceso

Cada vez que es invocado, Javadoc produce una documentación completa, por lo que no se puede conseguir resultados incrementales. Sin embargo, si una clase existente es utilizada en otra corrida de la documentación, la nueva instancia de Javadoc puede detectarlo e incluir la referencia existente.

Para crear las estructuras de herencia y de paquetes, Javadoc utiliza parte del compilador de Java, para obtener todas las referencias a otras clases, así como el nombre totalmente calificado para cada una de las clases y atributos. También verifica que se sigan las convenciones más básicas para el lenguaje (como el nombre de la clase, de los paquetes, etc.)

Una parte importante es que Javadoc utiliza clases de tipo **stub** y no las implementaciones finales de los métodos. Esto se hace para obtener la documentación completa aún antes de terminar todo el trabajo.

Javadoc también utiliza reglas implícitas de la estructura de Java, como por ejemplo, añadir los constructores vacíos aún cuando no se han especificado. Esto lo hace leyendo los archivos generados por el compilador (*.class*) en lugar de tan solo leer el archivo fuente proporcionado.

También se permite que en algunos contados casos, el archivo fuente contenga algunos errores o faltas a los estándares de Java, sin que se notifique al usuario. Un ejemplo de esto es la declaración de clases abstractas con métodos no abstractos.

Para construir la documentación de todas las clases implicadas Javadoc debe ser capaz de encontrarlas. Para ello se utiliza el algoritmo de Java Launcher, analizado en el anexo 1.2.

## Acerca de los *Doclets*

Los *Doclets* son formas de cambiar el contenido y el *look and feel* de los archivos generados por Javadoc. Aunque se incluye un *Doclet Default*, se incluyen maneras de modificarlo, o incluso de crear uno nuevo, a petición del usuario. En el caso del API de Java, se utilizó el *Doclet Default*, que incluye las siguientes clases:

Consiste de 3 paquetes que son:

**com.sun.tools.doclets**

**HtmlWriter.-** Clase que contiene el código para generar las etiquetas necesarias para dar formato al archivo HTML.

**HtmlDocWriter.-** Clase que hereda de *HtmlWriter*, e incluye cualquier etiqueta especializada para el HTML, como por ejemplo, los links a los encabezados y a los pies de página.

**Taglet.-** Una clase que realiza el papel de traductor entre los *tags* de Javadoc y los de HTML. Puede ser modificada o extendida para incluir *tags* personalizados.

**TagletManager.-** Es una clase que se ocupa de manejar una instancia de la clase Taglet. Contiene métodos para registrar y obtener *taglets*, así como de corrección de errores, que en este caso, tiene que ver con la inclusión de *tags* en lugares incorrectos. Un ejemplo de esto es que se intente utilizar el *tag @param* en la documentación de una clase, cuando sólo se debe de utilizar en la documentación de un método.

## com.sun.tools.doclets.standard

**Standard.-** Contiene el metodo Start() y por lo tanto es el encargado de manipular y controlar la generación de la documentación de los archivos y paquetes incluidos.

**PackageIndexWriter.-** Se encarga de generar el archivo overview-summary.html que contiene una lista de todos los archivos pasados por la línea de comandos.

**PackageIndexFrameWriter.-** Se encarga de generar le archivo overview-frame.html para desplegar la lista de todos los paquetes incluidos en la parte superior izquierda de la pantalla.

**PackageFrameWriter.-** Genera el archivo package-frame.html que contiene una lista de todas clases e interfaces incluidas en un paquete, desplegándolos en la parte inferior izquierda de la pantalla.

**FrameOutputWriter.-** Genera el archivo index.html que contiene los frames y la pagina de inicio de la documentación API.

**PackageWriter.-** Genera el archivo package-summary.html que contiene la lista de todas las clases e interfaces de un paquete, junto con sus explicaciones.

**ClassWriter.-** Genera un archivo html con el nombre de la clase, y contiene la documentación individual para ella.

**SingleIndexWriter.-** Genera el archivo de índice para los miembros de la clase cuando no se utiliza la opción `-splitindex`.

**SplitIndexWriter.-** Genera los múltiples archivos de índice para los miembros de la clase cuando se utiliza la opción `-splitindex`.

**TreeWriter.-** Genera el archivo overview-tree.index que contiene la lista de toda la jerarquía de clases.

**DeprecatedListWriter.-** Genera el archivo deprecatedlist.html que contiene la lista de todas las clases, métodos o interfaces que han sido desechados o eliminados de las nuevas implementaciones de la clase.

**ClassUseMapper, ClassUseWriter and PackageUseWriter.-** Generan un archivo html que contiene una lista de todas las clases o paquetes que hacen uso de la clase que se está documentando.

**Group.-** Genera un archivo HTML con todos los paquetes existentes, sin importar su relación, solo los lista alfabéticamente.

## com.sun.tools.doclets.standard.tags

**AbstractExecutableMemberTaglet.-** Un taglet abstracto que le da formato a un *tag* que se ejecuta durante la documentación.

**AbstractInlineTaglet.-** Un taglet abstracto que le da formato a un *tag* que se ejecuta en línea durante la documentación.

**DocRootTaglet.-** Un taglet que da formato al *tag* { @DocRoot }

**InheritDocTaglet.-** Un taglet que da formato al *tag* { @InheritDoc }

**ParamTaglet.-** Un taglet que da formato al *tag* @param.

**ReturnTaglet.-** Un taglet que da formato al *tag* @return.

**SeeTaglet.-** Un taglet que da formato al *tag* @see.

**SimpleTaglet.-** Un taglet que da formato a *tags* de un solo argumento como `@author`.

**ThrowsTaglet.-** Un taglet que da formato al *tag* `@throws`.

**ValueTaglet.-** Un taglet que da formato al *tag* `{@value}`.

Estas clases son las que Javadoc utiliza para el proceso de la documentación de las clases. Es importante señalar que estas clases son únicamente para el uso interno de Javadoc, por lo que no forman parte del API de Java y no es posible encontrar detalles e implementaciones de ellas.

## Semántica de la documentación en Javadoc

Hasta este punto, la descripción de los pasos ha sido totalmente interna, sin embargo, para que funcione correctamente, se tienen que seguir ciertas convenciones y semántica al momento de escribir los comentarios dentro del código fuente.

Los comentarios a documentar pueden incluirse justo antes de la declaración de clases, interfaces, métodos, constructores o atributos.

Para indicar a Javadoc que los comentarios se deben de incluir entre los signos `/** */`, como en el siguiente ejemplo:

```
/** Este es un comentario de JavaDoc.  
 * @author KingCrimson  
 */
```

Todos los comentarios deben de escribirse **exactamente antes** de la parte a la que se quieren referir.

La primera frase de los comentarios se toma desde que comienza el bloque con los dos asteriscos y hasta el primer punto, tag o tabulador. Esta primera frase se copia en la página de *summary*, por lo que debe de consistir de una descripción breve de la función que se espera del miembro que se está comentando.

Javadoc también acepta el uso de etiquetas HTML, aunque deben de utilizarse con cuidado, puesto que no siguen la misma estructura de Javadoc y pueden modificar el *layout*, de la página.

Javadoc también tiene la habilidad de crear comentarios para clarificar ciertos aspectos aparentemente olvidados por el programador. Existen 3 casos para los cuales se incluyen comentarios no establecidos:

1. **Creación automática de comentarios faltantes.-** Se crea la documentación para las partes críticas que no fueron comentadas, como por ejemplo, la omisión de los correspondientes tags `@param` o `@return`, aún cuando si existan parámetros y objetos de regreso. En este caso, únicamente se mencionan en su apartado correspondiente.
2. **Herencia explícita de código.-** Cuando se utiliza el tag `{@inheritDoc}` la descripción de los comentarios son copiados a donde se encuentre el tag mencionado.

3. **Herencia implícita de código.**- Cuando mediante la estructura de clases de Java se encuentra que ciertas partes son heredadas de una superclase, pero aún tienen aportaciones vitales, como en los siguientes casos:
  - a. Cuando un método en una clase hace *override* de otro método en una superclase.
  - b. Cuando un método de una interfaz hace *override* de otro método en una superinterface.
  - c. Cuando un método implementa un método descrito en una interfaz.

Los tags disponibles se encuentran dentro del Anexo 1.3.

De esta manera, se describe de forma general el funcionamiento, alcance e implementación de Javadoc, concluyendo (aunque no de manera definitiva) la primera etapa del proyecto.

## Anexo 1.1 Opciones modificadoras de Javadoc

### -1.1

Esta función ha sido removida en la nueva versión 1.4. Sirve para hacer documentación compatible con la versión 1.1 en la que no existen clases anidadas.

#### **-author**

Incluye el nombre del autor en la documentación generada, a partir del *tag* @author.

#### **-bootclasspath** *classpathList*

Especifica en donde se encuentran las clases de la plataforma de Java. Esta parte es la que Java utilizará para encontrar las clases en la hora de la compilación.

#### **-bottom** *text*

Especifica un texto que será incluido en la parte de debajo de cada archivo obtenido.

#### **-breakiterator**

Cambia la forma de obtener la primer oración de cada bloque de comentarios, misma que se utilizará en el resumen.

#### **-charset**

Especifica el conjunto de caracteres HTML que se utilizará para la documentación.

#### **-classpath** *classpathlist*

Especifica la localización de las clases referenciadas por los archivos fuente.

#### **-d** *directory*

Especifica el directorio en donde se colocará los archivos resultantes.

#### **-docencoding** *name*

Especifica el encoding para los archivos HTML generados. Parecido a `-charset`.

#### **-docfilessubdirs**

En caso de una copia hacia otro directorio, este comando provoca que la copia se haga de manera “profunda” es decir, con sus subdirectorios.

#### **-doclet** *class*

Especifica el archivo class que contiene el *doclet* para la generación de los documentos. Este archivo debe de contener el método `start()` (método raíz).

#### **-docletpath**

Especifica el lugar en donde se encuentra el archivo class que contiene el *doclet*.

#### **-doctitle** *title*

Especifica un texto que servirá como titulo en la parte superior central del archivo `overview-summary.html`.

#### **-encoding** *name*

Especifica el tipo de *encoding* de los archivos fuente, como EUCJIS.

#### **-exclude** *package1:package2*

Especifica los paquetes que no se tomarán en cuenta en el proceso de documentación.

#### **-extdirs** *dirlist*

Especifica el lugar en donde residen las clases de Extensión (*i??*)

#### **-footer** *footer*

Especifica un texto que será colocado en la parte inferior de cada archivo generado, a la derecha de la barra de navegación más baja.

#### **-group** *groupheading package1:package2*

Separa todos los paquetes en grupos distintivos dentro del archivo `overview.html`

#### **-header** *header*

Especifica un encabezado que será colocado en todos los archivos generados, que estará a la derecha de la barra de navegación superior.

**-help**

Despliega la ayuda en línea de Javadoc, que es esta especificación de los parámetros posibles.

**-helpfile** *path/filename*

Especifica un archivo de ayuda distinto al generado por defecto, que es al que apunta el link HELP en la parte superior de los archivos html.

**-J**

Cambia alguna variable de la maquina virtual que esta corriendo Javadoc. No admite espacio entre el modificador y su argumento.

**-link** *extDocument*

Especifica un archivo html externo que contiene clases ya documentadas y que se quiere añadir a la nueva corrida de Javadoc que se quiere crear.

**-linkoffline** *extDocument*.

De manera similar a `-link`, pero especificando que el contenido está offline.

**-linksource**

Crea un archivo HTML para cada archivo fuente y los junta dentro de la documentación estandar, mostrando todos sus métodos, atributos y clases, sin importar si son privados o protegidos.

**-locale**

Especifica un nombre contenido dentro de `java.util.Locale` en la que se escribirá la documentación.

**-nocomment**

Elimina todos los cuerpos de comentarios, generando solamente un archivo con los nombres de métodos, clases y atributos.

**-nodeprecated**

Elimina la generación de API de tipo *deprecated*.

**-nodeprecatedlist**

Elimina la generación de API del tipo *deprecated*, y además, elimina el link a esta lista dentro de la documentación final.

**-nohelp**

Elimina el link al archivo de ayuda dentro de la barra de navegación superior.

**-noindex**

Omite la generación de un archivo `index.html`

**-nonavbar**

Previene la generación de la barra de navegación, tanto superior como inferior.

**-noqualifier** *all / package1:package2*

Elimina la construcción de la ruta completamente calificada de los paquetes.

**-nosince**

Omite el texto generado por el *tag @since*.

**-notree**

Omite la jerarquía de clases o interfaces en los archivos de documentación generados.

**-overview** *path*

Especifica que archivo debe ser utilizado como `overview.html` en lugar de generar uno nuevo.

**-package**

Muestra solo los paquetes y las clases protegidas, publicas y miembros.

**-private**

Muestra todas las clases y miembros.

**-protected**

Muestra solo las clases y miembros públicos y protegidos. Es la opción por defecto.

**-public**

Muestra solo las clases y miembros públicos.

**-quiet**

Elimina todos los mensajes que no sean de error o de advertencia.

**-serialwarn**

Genera advertencias en tiempo de compilación para mostrar *tags* que no hayan sido escritos pero que sean necesarios. Un ejemplo es un método con 3 parámetros y solo 2 @param.

**-source**

Necesario para hacer que Javadoc maneje invariantes de tipo 1.4

**-sourcepath** *sourcepath*

Especifica el lugar en donde se encuentran los archivos fuentes a procesar.

**-splitindex**

Divide el index en diferentes archivos, alfabéticamente, un archivo por letra.

**-stylesheetfile** *path*

Especifica un archivo .css distinto al generado por defecto.

**-subpackages** *package1:package2*

Especifica que los paquetes a documentar también incluyen de manera recursiva a los subpaquetes de cada uno.

**-tag**

Habilita a Javadoc para interpretar un simple *tag* de manera personalizada.

**-taglet** *class*

Especifica la clase que contiene los *taglets* utilizados en la generación de la documentación.

**-tagletpath**

Especifica en donde se encuentra la clase que contiene los *taglets*.

**-use**

Crea un archivo use.html para cada clase o paquete documentado.

**-verbose**

Provee mensajes detallados del proceso que se ejecuta al correr Javadoc.

**-version**

Incluye el texto generado por el *tag* @version

**-windowtitle** *title*

Especifica el texto que será incluido como Título de la pagina de documentación, es decir, dentro de los tags de HTML <title></title>

## Anexo 1.2 Algoritmo Java Launcher para encontrar clases implicadas.

Para iniciar una nueva Máquina Virtual se utiliza `JavaLauncher.java`, que para funcionar correctamente necesita de todas las clases implicadas, las cuales busca en el siguiente orden:

**Bootstrap classes.-** Son las clases que comprenden la plataforma de Java, incluidas, entre otros archivos, en `rt.jar` y en el directorio `jre/lib`.

**Extensión classes.-** Son las clases que utilizan el mecanismo de herencia de Java. También están incluidas en archivos `.jar`. Todos los archivos que se encuentran dentro del directorio `jre/lib/ext` se asumen como archivos que contienen clases de *extensión*, por lo que una clase que no esté contenida en uno de estos archivos es ignorada. Si una clase se encuentra en dos archivos `.jar` distintos, se toma como indefinida.

**User classes.-** Clases definidas por el usuario.

Las clases de *bootstrap* y *extensión* se buscan dentro de la variable de entorno *classpath*. De esta manera, al compilar únicamente se le tiene que pasar como argumento la localización de las clases de usuario, generalmente en el mismo directorio de donde se está corriendo el compilador.

Tanto `Javac` como `Javadoc` necesitan varias clases para poder funcionar. Esto es por que el archivo fuente que reciben puede hacer referencias a clases nativas de Java o clases de usuario, por lo que tanto `Javac` como `Javadoc` tienen que resolver esas referencias. Tales referencias pueden aparecer en forma de archivos `.class`, `.java`, o ambas.

Las clases que se encuentran en `tools.jar` sólo se utilizan para ejecutar `Javac` o `Javadoc`. Estas clases no son utilizadas para resolver las referencias antes mencionadas, a menos de que sean incluidas explícitamente en el *classpath* del usuario.

Si por alguna razón el programador desea incluir clases distintas a las implementadas por la plataforma de Java, puede hacerlo mediante la opción `-bootclasspath`. Sin embargo, el uso de este modificador no cambia las clases de `tools.jar` utilizadas para correr `Javac` o `Javadoc`.

En cuanto a las diferencias entre `Javac` y `Javadoc`, el segundo sólo utiliza los archivos fuente (aunque realiza algunas validaciones con `Javac`). El primero solo se enfoca a los archivos `.class` (para resolver las dependencias) y puede discriminar si alguno de estos archivos se refiere a una versión no actualizada, para así recompilarlo.

Una vez que ya están compiladas, una clase debe de ser cargada por un *ClassLoader*, que tiene políticas de seguridad asociadas.

A nivel de usuario, para cargar una clase solo se tiene que crear una referencia hacia ella. Sin embargo, esto invoca a un *ClassLoader* interno que aplica ciertas políticas de

seguridad. Por defecto, si no se especifica algún manejador de seguridad, todas las clases son permitidas y no existen restricciones de ningún tipo.

Si se define un nuevo manejador, Java les da completa libertad a las clases que son hijas de la clase cargada, y restricciones básicas a las demás. Por supuesto, se permite definir de manera personalizada las restricciones.

## Anexo 1.3 Tags disponibles de Javadoc.

### **@author**

Utilizado para especificar el autor de dicho archive.

### **{@docRoot}**

Representa el *path* relativo, que todos los demás archivos tendrán en cuenta, tomando éste como el directorio raíz.

### **@deprecated**

Especifica que el API incluido no deberá de utilizarse en las nuevas versiones, aunque puede funcionar, pero que no es recomendado.

### **@exception**

Sinónimo del *tag* @throws.

### **{@inheritDoc}**

Copia la documentación de la clase hija “mas cercana” o interfaz implementable, en el lugar en donde se encuentre este *tag*.

### **{@link packageclass#memberclass}**

Crea un link a la clase determinada como argumento.

### **{@linkplain packageclass#memberclass}**

Similar al anterior, solo que éste muestra el link como texto sin formato.

### **@param parameter-name description**

Añade el parámetro a la sección de parámetros del método.

### **@return description**

Añade el objeto retorno a la sección de Retorno. Únicamente válido en métodos.

### **@see**

Similar a @link.

### **@serial**

Añade un comentario a un campo serializable.

### **@serialData DataDescription**

Añade un comentario a los datos de forma serializable.

### **@since**

Añade una especificación aclarando a partir de cual versión se utiliza el objeto documentado.

### **@throws**

Añade un comentario a la sección de excepciones y *throws*.

### **{@value}**

Despliega el valor de todas las constantes declaradas.

### **@version**

Añade la versión del archivo que se esta documentando.