

[\(http://baeldung.com\)](http://baeldung.com)

Introducción a Javadoc

Última modificación: 24 de febrero de 2018

por [baeldung](http://www.baeldung.com/author/baeldung/) (<http://www.baeldung.com/author/baeldung/>)

[Java](http://www.baeldung.com/category/java/) (<http://www.baeldung.com/category/java/>) +

Acabo de anunciar los nuevos módulos de *Spring 5* en REST With Spring:

[-> COMPRUEBA EL CURSO \(/rest-with-spring-course#new-modules\)](#)

1. Información general

Una buena documentación API es uno de los muchos factores que contribuyen al éxito general de un proyecto de software. Afortunadamente, todas las versiones modernas del JDK proporcionan la herramienta Javadoc, para generar documentación API a partir de los comentarios presentes en el código fuente.

Prerrequisitos:

1. JDK 1.4 (se recomienda JDK 7+ para la última versión del plugin Maven Javadoc)
2. La carpeta JDK / *bin* añadida a la *variable de entorno PATH*
3. (Opcional) un IDE que con herramientas incorporadas

2. Comentarios de Javadoc

Comencemos con los comentarios.

La estructura de comentarios de Javadoc es muy similar a un comentario regular de varias líneas, pero la diferencia clave es el asterisco adicional al principio:

```
1 // This is a single line comment
2
3 /*
4  * This is a regular multi-line comment
5  */
6
7 /**
8  * This is a Javadoc
9  */
```

Los comentarios de estilo de Javadoc también pueden contener etiquetas HTML.

2.1. Formato Javadoc

Los comentarios de Javadoc pueden colocarse sobre cualquier clase, método o campo que deseemos documentar.

Estos comentarios se componen comúnmente de dos secciones:

1. La descripción de lo que estamos comentando
2. Las etiquetas de bloque independientes (marcadas con el símbolo " @ ") que describen metadatos específicos

Usaremos algunas de las etiquetas de bloque más comunes en nuestro ejemplo. Para obtener una lista completa de las etiquetas de bloque, visite la guía de referencia (<https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html>).

2.2. Javadoc a nivel de clase

Echemos un vistazo a cómo se vería un comentario de Javadoc a nivel de clase:

```
1  /**
2   * Hero is the main entity we'll be using to . . .
3   *
4   * Please see the {@link com.baeldung.javadoc.Person} class for true identity
5   * @author Captain America
6   *
7   */
8   public class SuperHero extends Person {
9       // fields and methods
10  }
```

Tenemos una breve descripción y dos etiquetas de bloque diferentes, independientes y en línea:

- Las etiquetas independientes aparecen después de la descripción con la etiqueta como la primera palabra en una línea, por ejemplo, la etiqueta `@author`
- **Las etiquetas en línea pueden aparecer en cualquier lugar y están rodeadas con llaves**, por ejemplo, la etiqueta `@link` en la descripción

En nuestro ejemplo, también podemos ver dos tipos de etiquetas de bloque que se utilizan:

- `{@link}` proporciona un enlace en línea a una parte referenciada de nuestro código fuente
- `@author` el nombre del autor que agregó la clase, el método o el campo que se comentó

2.3. Javadoc en el nivel de campo

También podemos usar una descripción sin etiquetas de bloqueo como esta dentro de nuestra clase *SuperHero*:

```
1  /**
2   * The public name of a hero that is common knowledge
3   */
4   private String heroName;
```

Los campos privados no tendrán generado Javadoc a menos que pasamos explícitamente la opción `-private` al comando Javadoc.

2.4. Javadoc en el nivel de método

Los métodos pueden contener una variedad de etiquetas de bloque Javadoc.

Echemos un vistazo a un método que estamos usando:

```
1  /**
2   * <p>This is a simple description of the method. . .
3   * <a href="http://www.supermanisthegreatest.com (http://www.supermanisthegreatest.com)">Superman!</a>
4   * </p>
5   * @param incomingDamage the amount of incoming damage
6   * @return the amount of health hero has after attack
7   * @see <a href="http://www.link_to_jira/HERO-402 (http://www.link_to_jira/HERO-402)">HERO-402</a>
8   * @since 1.0
9   */
10 public int successfullyAttacked(int incomingDamage) {
11     // do things
12     return 0;
13 }
```

El método *successfullyAttacked* contiene una descripción y numerosas etiquetas de bloques independientes.

Hay muchas etiquetas de bloques para ayudar a generar la documentación adecuada y podemos incluir todo tipo de diferentes tipos de información. Incluso podemos utilizar etiquetas HTML básicas en los comentarios.

Repasemos las etiquetas que encontramos en el ejemplo anterior:

- *@param* proporciona cualquier descripción útil sobre el parámetro de un método o entrada que debería esperar
- *@return* proporciona una descripción de lo que un método puede o puede devolver
- *@see* generará un enlace similar a la etiqueta *!@link!*, pero más en el contexto de una referencia y no en línea
- *@since* especifica qué versión se agregó la clase, el campo o el método al proyecto
- *@version* especifica la versión del software, comúnmente utilizada con las macros *%I%* y *%G%*
- *@throws* se usa para explicar mejor los casos en que el software esperaría una excepción
- *@deprecated* da una explicación de por qué el código fue desaprobado, cuándo puede haber quedado obsoleto y cuáles son las alternativas

Aunque ambas secciones son técnicamente opcionales, necesitaremos al menos una para que la herramienta Javadoc genere algo significativo.

3. Generación Javadoc

Para generar nuestras páginas Javadoc, querremos echar un vistazo a la herramienta de línea de comandos que viene con el JDK y el plugin Maven.

3.1. Herramienta de línea de comandos de Javadoc

La herramienta de línea de comandos de Javadoc es muy poderosa pero tiene cierta complejidad asociada.

Ejecutar el comando *javadoc* sin ninguna opción o parámetro dará como resultado un error y los parámetros de salida que espera.

Necesitaremos al menos especificar para qué paquete o clase queremos que se genere la documentación.

Vamos a abrir una línea de comando y navegar al directorio del proyecto.

Suponiendo que las clases están todas en la carpeta *src* en el directorio del proyecto:

```
1 | user@baeldung:~$ javadoc -d doc src\*
```

Esto generará documentación en un directorio llamado *doc* como se especifica con el indicador *-d*. Si existen múltiples paquetes o archivos, tendremos que proporcionarlos a todos.

Utilizar un IDE con la funcionalidad incorporada es, por supuesto, más fácil y generalmente recomendado.

3.2. Javadoc con Maven Plugin

También podemos hacer uso del plugin Maven Javadoc:

```
1 | <build>
2 |   <plugins>
3 |     <plugin>
4 |       <groupId>org.apache.maven.plugins</groupId>
5 |       <artifactId>maven-javadoc-plugin</artifactId>
6 |       <version>3.0.0</version>
7 |       <configuration>
8 |         <source>1.8</source>
9 |         <target>1.8</target>
10 |       </configuration>
11 |     </plugin>
12 |   </plugins>
13 | </build>
```

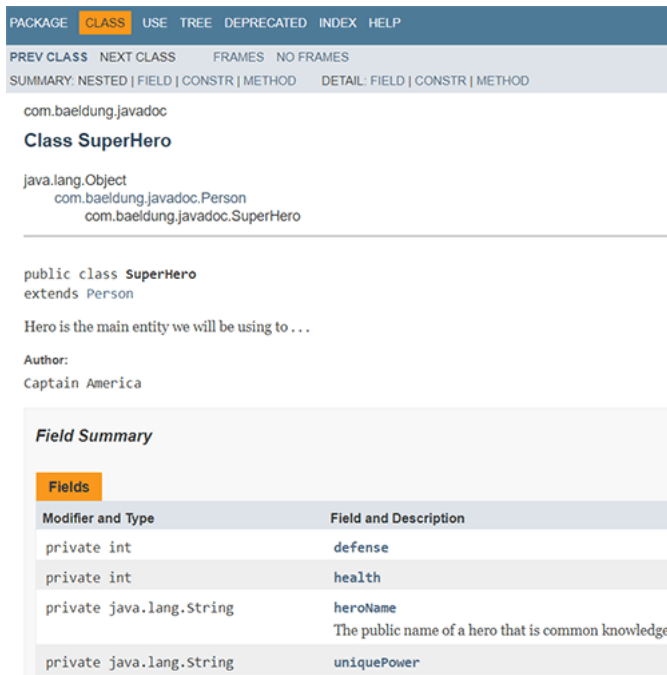
En el directorio base del proyecto, ejecutamos el comando para generar nuestros Javadocs en un directorio en *target \ site*:

```
1 | user@baeldung:~$ mvn javadoc:javadoc
```

El plugin Maven

(<https://search.maven.org/#search%7Cgav%7C1%7Cg%3A%22org.apache.maven.plugins%22%20AND%20a%3A%22maven-javadoc-plugin%22>) es muy poderoso y facilita la generación de documentos complejos sin problemas.

Veamos ahora cómo se ve una página Javadoc generada:



PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

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

com.baeldung.javadoc

Class SuperHero

java.lang.Object
com.baeldung.javadoc.Person
com.baeldung.javadoc.SuperHero

public class **SuperHero**
extends **Person**

Hero is the main entity we will be using to ...

Author:
Captain America

Field Summary

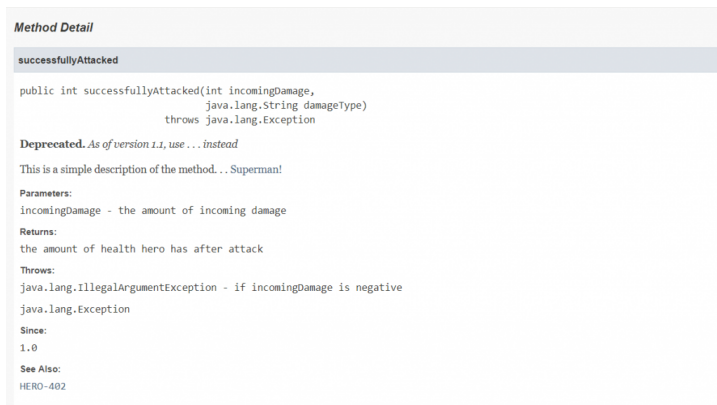
Modifier and Type	Field and Description
private int	defense
private int	health
private java.lang.String	heroName The public name of a hero that is common knowledge
private java.lang.String	uniquePower

([http://www.baeldung.com/wp-](http://www.baeldung.com/wp-content/uploads/2018/01/overview.png)

[content/uploads/2018/01/overview.png](http://www.baeldung.com/wp-content/uploads/2018/01/overview.png))

Podemos ver una vista en árbol de las clases que se extiende a nuestra clase *SuperHero*. Podemos ver nuestra descripción, campos y método, y podemos hacer clic en enlaces para obtener más información.

Una vista detallada de nuestro método se ve así:



Method Detail

successfullyAttacked

```
public int successfullyAttacked(int incomingDamage,
    java.lang.String damageType)
    throws java.lang.Exception
```

Deprecated. As of version 1.1, use ... instead

This is a simple description of the method. ... Superman!

Parameters:
incomingDamage - the amount of incoming damage

Returns:
the amount of health hero has after attack

Throws:
java.lang.IllegalArgumentException - if incomingDamage is negative
java.lang.Exception

Since:
1.0

See Also:
HERO-402

([http://www.baeldung.com/wp-](http://www.baeldung.com/wp-content/uploads/2018/01/ss_javadoc-1024x579.png)

[content/uploads/2018/01/ss_javadoc-1024x579.png](http://www.baeldung.com/wp-content/uploads/2018/01/ss_javadoc-1024x579.png))

3.3. Etiquetas personalizadas de Javadoc

Además de usar etiquetas de bloque predefinidas para formatear nuestra documentación, **también podemos crear etiquetas de bloque personalizadas.**

Para hacerlo, solo necesitamos incluir una opción `-tag` en nuestra línea de comando Javadoc en el formato de `<nombre-etiqueta>: <ubicaciones-permitido>: <encabezado>`.

Para crear una etiqueta personalizada llamada `@location` allowed anywhere, que se muestra en el encabezado "Notable Locations" en nuestro documento generado, debemos ejecutar:

```
1 | user@baeldung:~$ javadoc -tag location:a:"Notable Locations:" -d doc src\*
```

Para usar esta etiqueta, podemos agregarla a la sección de bloque de un comentario de Javadoc:

```
1  /**
2   * This is an example...
3   * @location New York
4   * @returns blah blah
5   */
```

El plugin Maven Javadoc es lo suficientemente flexible como para permitir también las definiciones de nuestras etiquetas personalizadas en nuestro *pom.xml*.

Para configurar la misma etiqueta anterior para nuestro proyecto, podemos agregar lo siguiente a la sección *<etiquetas>* de nuestro complemento:

```
1  ...
2  <tags>
3      <tag>
4          <name>location</name>
5          <placement>a</placement>
6          <head>Notable Places:</head>
7      </tag>
8  </tags>
9  ...
```

De esta forma, podemos especificar la etiqueta personalizada una vez, en lugar de especificarla cada vez.

4. Conclusión

Este tutorial de introducción rápida cubrió cómo escribir Javadocs básicos y generarlos con la línea de comandos de Javadoc.

Una forma más fácil de generar la documentación sería utilizar cualquier opción IDE incorporada o incluir el complemento Maven en nuestro archivo *pom.xml* y ejecutar los comandos apropiados.

Las muestras de código, como siempre, pueden encontrarse en GitHub (<https://github.com/eugenp/tutorials/tree/master/core-java>).

Acabo de anunciar los nuevos módulos de Spring 5 en REST With Spring:

>> VERIFIQUE LAS LECCIONES (/rest-with-spring-course#new-modules)



(<http://www.baeldung.com/wp-content/uploads/2016/05/baeldung-rest-post-footer-main-1.2.0.jpg>)



(<http://www.baeldung.com/wp-content/uploads/2016/05/baeldung-rest-post-footer-icn-1.0.0.png>)

Aprendiendo a "Construir tu API con Spring "

>> Obtener el eBook



Descargar el libro electrónico

¿Construir una API REST con
Spring 4?

[Descargar](#)

CATEGORÍAS

PRIMAVERA ([HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/](http://www.baeldung.com/category/spring/))
DESCANSO ([HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/](http://www.baeldung.com/category/rest/))
JAVA ([HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/](http://www.baeldung.com/category/java/))
SEGURIDAD ([HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/](http://www.baeldung.com/category/security-2/))
PERSISTENCIA ([HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/](http://www.baeldung.com/category/persistence/))
JACKSON ([HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/](http://www.baeldung.com/category/jackson/))
HTTPCLIENT ([HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/](http://www.baeldung.com/category/http/))
KOTLIN ([HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/](http://www.baeldung.com/category/kotlin/))

SERIE

TUTORIAL "VOLVER A LO BÁSICO" DE JAVA ([HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL](http://www.baeldung.com/java-tutorial))
JACKSON JSON TUTORIAL ([HTTP://WWW.BAELDUNG.COM/JACKSON](http://www.baeldung.com/jackson))
TUTORIAL DE HTTPCLIENT 4 ([HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE](http://www.baeldung.com/httpclient-guide))
REST CON SPRING TUTORIAL ([HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/](http://www.baeldung.com/rest-with-spring-series/))
TUTORIAL DE SPRING PERSISTENCE ([HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/](http://www.baeldung.com/persistence-with-spring-series/))
SEGURIDAD CON SPRING ([HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING](http://www.baeldung.com/security-spring))

ACERCA DE

ACERCA DE BAELDUNG ([HTTP://WWW.BAELDUNG.COM/ABOUT/](http://www.baeldung.com/about/))
LOS CURSOS ([HTTP://COURSES.BAELDUNG.COM](http://courses.baeldung.com))
TRABAJO DE CONSULTORÍA ([HTTP://WWW.BAELDUNG.COM/CONSULTING](http://www.baeldung.com/consulting))
META BAELDUNG ([HTTP://META.BAELDUNG.COM/](http://meta.baeldung.com/))
EL ARCHIVO COMPLETO ([HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE](http://www.baeldung.com/full_archive))
ESCRIBIR PARA BAELDUNG ([HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES](http://www.baeldung.com/contribution-guidelines))
CONTACTO ([HTTP://WWW.BAELDUNG.COM/CONTACT](http://www.baeldung.com/contact))
INFORMACIÓN DE LA COMPAÑÍA ([HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO](http://www.baeldung.com/baeldung-company-info))
TÉRMINOS DE SERVICIO ([HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE](http://www.baeldung.com/terms-of-service))
POLÍTICA DE PRIVACIDAD ([HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY](http://www.baeldung.com/privacy-policy))
EDITORES ([HTTP://WWW.BAELDUNG.COM/EDITORS](http://www.baeldung.com/editors))
KIT DE MEDIOS (PDF) ([HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF](https://s3.amazonaws.com/baeldung.com/baeldung+-+media+kit.pdf))