How to write JavaDoc Comments

Vishnu Kotrajaras, PhD
Modified from Sun's document

Where to place?

- Immediately ahead of declarations for any class, interface, method, constructor, or field.
- Documentation comments placed in the body of a method are ignored.
- Only one documentation comment per declaration statement is recognized by the Javadoc tool.

What does it look like?

```
/**
*This is the typical format of
*…..
*/
```

To save space you can put a comment on one line:

```
/** This comment takes up only one line. */
```

The comment text should use HTML entities and can use HTML tags. But don't use HTML heading tags such as <H1> and <H2>, because Javadoc will get confused.

Careful about positioning

- A common mistake is to put an import statement between the class comment and the class declaration. Avoid this, as the Javadoc tool will ignore the class comment.

```
/**
*This is the class comment for the class Whatever.
*/
import com.sun; // MISTAKE
public class Whatever { }
```

Comment can be divided to 2 regions

```
/**
* This sentence would hold the main
*description for this doc comment.
* @see java.lang.Object
*/
```

Main description

Tag section

The argument to a tag can span multiple lines. There can be any number of tags

The main description cannot continue after the tag section begins.

First Sentence of JavaDoc comment

- The first sentence of each doc comment should be a summary sentence
- The Javadoc tool copies this first sentence to the appropriate member, class/interface or package summary

First Sentence of JavaDoc comment (cont.)

- This sentence ends at the first period that is followed by a blank, tab, or line terminator, or at the first tag (as defined below). For example, this first sentence ends at "Prof.":

```
/**
* This is a simulation of Prof. Knuth's MIX computer.
*/
```

First Sentence of JavaDoc comment (cont2.)

- However, you can work around this by typing an HTML meta-character such as "&" or "<" immediately after the period, such as:

```
/**
* This is a simulation of Prof. Knuth's MIX computer.
*/
```

or

```
/**
* This is a simulation of Prof.<!-- --> Knuth's MIX computer.
*/
```

First Sentence of JavaDoc comment (cont3.)

- In particular, write summary sentences that distinguish overloaded methods from each other. For example:

```
/**
 * Class constructor.
*/
foo() { ...
/**
* Class constructor specifying number of objects to create.
*/
foo(int n) { ...
```

Block tags and in-line tags

- block tags appear as @tag
  - must appear at the beginning of a line, ignoring leading asterisks, white space, and separator (/**).
  - This means you can use the @ character elsewhere in the text and it will not be interpreted as the start of a tag.
  - Each block tag has associated text:
    - any text following the tag up to, but not including, either the next tag, or the end of the doc comment. This associated text can span multiple lines.
- in-line tags appears within curly braces, as {@tag}.

Example of block and inline tag

```
/**
* @deprecated As of JDK 1.1, replaced by
   *{@link#setBounds(int,int,int,int)}
*/
```

Example of HTML tag in comment

```
/**
*This is a <b>doc</b> comment.
* @see java.lang.Object
*/
```

bold

less-than (<) and greater-than (>) symbols should be written < and >.

the ampersand (&) should be written &

Declaration with multiple fields

```
/**
*The horizontal and vertical distances of point (x,y)
*/
public int x, y;
```

- The Javadoc tool will copy one comment for every variable.

public int **x**

The horizontal and vertical distances of point (x,y)

public int **y**

The horizontal and vertical distances of point (x,y)

- So we should write their comments separately.

Automatic Copying of Method Comments

- Constructors, fields and nested classes do not inherit doc comments.
- **Automatically inherit comment to fill in missing text**
  - When a main description, or @return, @param or @throws tag is missing from a method comment, the Javadoc tool copies the corresponding main description or tag comment from the method it overrides or implements (if any).
    - when a @param tag for a particular parameter is missing, then the comment for that parameter is copied from the method further up the inheritance hierarchy.
    - When a @throws tag for a particular exception is missing, the @throws tag is copied *only if that exception is declared*.

  - This behavior contrasts with version 1.3 and earlier, where the presence of any main description or tag would prevent all comments from being inherited.
- **Explicitly inherit comment with {@inheritDoc} tag**
  - Insert the inline tag {@inheritDoc} in a method main description or @return, @param or @throws tag comment -- the corresponding inherited main description or tag comment is copied into that spot.
- The source file for the inherited method need only be on the path specified by -sourcepath for the doc comment to actually be available to copy. Neither the class nor its package needs to be passed in on the command line.
  - This contrasts with 1.3.x and earlier releases, where the class had to be a documented class

Inherit from classes and interfaces

- When a method in a class overrides a method in a superclass
- When a method in an interface overrides a method in a superinterface
  - In these two cases, for method overrides, the Javadoc tool generates a subheading "Overrides" in the documentation for the overriding method, with a link to the method it is overriding, whether or not the comment is inherited.
- When a method in a class implements a method in an interface
  - In this case, the Javadoc tool generates a subheading "Specified by" in the documentation for the overriding method, with a link to the method it is implementing. This happens whether or not the comment is inherited.

Algorithm for Inheriting Method Comments

1. Look in each directly implemented (or extended) interface in the order they appear following the word implements (or extends) in the method declaration. Use the first doc comment found for this method.
2. If step 1 failed to find a doc comment, recursively apply this entire algorithm to each directly implemented (or extended) interface, in the same order they were examined in step 1.
3. If step 2 failed to find a doc comment and this is a class other than Object (not an interface):
   a. If the superclass has a doc comment for this method, use it.
   b. If step 3a failed to find a doc comment, recursively apply this entire algorithm to the superclass.

Tags

- **@author** *name-text*
  - Adds an "Author" entry with the specified *name-text* to the generated docs when the -author option is used.
  - A doc comment may contain multiple @author tags.
  - can only be applied at the overview, package and class level
  - You can specify one name per @author tag
    - the Javadoc tool inserts a comma (,) and space between names.
  - or multiple names per tag.
    - the entire text is simply copied to the generated document without being parsed. Therefore, you can use multiple names per line if you want a localized name separator other than comma.

- **@deprecated** *deprecated-text*
  - Adds a comment indicating that this API should no longer be used (even though it may continue to work). The Javadoc tool moves the *deprecated-text* ahead of the main description, placing it in italics and preceding it with a bold warning: "Deprecated". This tag is valid in all doc comments: overview, package, class, interface, constructor, method and field.
  - The first sentence of *deprecated-text* should at least tell the user when the API was deprecated and what to use as a replacement. The Javadoc tool copies just the first sentence to the summary section and index. Subsequent sentences can also explain why it has been deprecated.
  - You should include a {@link} tag (for Javadoc 1.2 or later) that points to the replacement API:

```
/**
*@deprecated As of JDK 1.1, replaced by {@link #setBounds(int,int,int,int)}
 */
```

- **{@docRoot}**
  - Represents the relative path to the generated document's (destination) root directory from any generated page.
  - It is useful when you want to include a file, such as a copyright page or company logo, that you want to reference from all generated pages.
  - Linking to the copyright page from the bottom of each page is common.
  - This {@docRoot} tag can be used both on the command line and in a doc comment.
  - This tag is valid in all doc comments: overview, package, class, interface, constructor, method and field, including the text portion of any tag (such as @return, @param and @deprecated).

1. On the command line, where the header/footer/bottom are defined:

```
javadoc -bottom '<a
    href="{@docRoot}/copyright.html">Copyright</a>'
```

2. In a doc comment:

```
/**
*See the <a href="{@docRoot}/copyright.html">Copyright</a>.
*/
```

<a href="{@docRoot}/copyright.html">

would resolve to: <a href="../../copyright.html"> for java/lang/Object.java and

<a href="../../../copyright.html"> for java/lang/ref/Reference.java

- **{@inheritDoc}**
  - **Note: This feature is broken in 1.4.0, but fixed in 1.4.1**
  - Inherits (copies) documentation from the "nearest" inheritable class or implementable interface into the current doc comment at this tag's location.
  - This allows you to write more general comments higher up the inheritance tree, and to write around the copied text.
  - This tag is valid only in these places in a doc comment:
    - In the main description block of a method. In this case, the main description is copied from a class or interface up the hierarchy.
    - In the text arguments of the @return, @param and @throws tags of a method. In this case, the tag text is copied from the corresponding tag up the hierarchy.

- **{@link** *package.class#member label***}**
  - Inserts an in-line link with visible text *label* that points to the documentation for the specified package, class or member name of a referenced class.
  - This tag is valid in all doc comments: overview, package, class, interface, constructor, method and field, including the text portion of any tag (such as @return, @param and @deprecated).
  - There is no limit to the number of {@link} tags allowed in a sentence.
  - This tag is very simliar to @see -- both require the same references and accept exactly the same syntax for *package.class#member* and *label*. The main difference is that {@link} generates an in-line link rather than placing the link in the "See Also" section. Also, the {@link} tag begins and ends with curly braces to separate it from the rest of the in-line text.
  - If you need to use "}" inside the label, use the HTML entity notation }

Link example

- Use the {@link #getComponentAt(int, int) getComponentAt} method.

label

becomes

Use the getComponentAt method.

On the web, it'll be

Use the getComponentAt method.

- **{@linkplain** *package.class#member label* **}**
  - Identical to {@link}, except the link's label is displayed in plain text than code font. Useful when the label is plain text.
  - Example:

Refer to {@linkplain add() the overridden method}.

  - This would display as:

Refer to the overridden method.

- **@param** *parameter-name description*
  - Adds a parameter to the "Parameters" section. The description may be continued on the next line.
  - Additional spaces can be inserted between the name and description so that the descriptions line up in a block.
  - This tag is valid only in a doc comment for a method or constructor.
  - @param tag is "required" (by convention) for every parameter, even when the description is obvious.
    - The @param tag is followed by the name (not data type) of the parameter, followed by a description of the parameter. The first noun in the description is the data type of the parameter, except if the parameter is primitive- > we can omit this type.
    - Dashes or other punctuation should not be inserted before the description, as the Javadoc tool inserts one dash.
      - The data type starts with a lowercase letter.
      - The description begins with a lowercase letter if it is a phrase (contains no verb), or an uppercase letter if it is a sentence.
      - End the phrase with a period only if another phrase or sentence follows it.

@param example

```
* @param ch          the character to be tested
* @param observer    the image observer to be
                     notified
```

- Do not bracket the name of the parameter after the @param tag with <code>...</code> since Javadoc 1.2 and later automatically do this.

@param example 2

- When writing a phrase, do not capitalize and do not end with a period:

```
@param x  the x-coordinate, measured in pixels
```

- When writing a phrase followed by a sentence, do not capitalize the phrase, but end it with a period to distinguish it from the start of the next sentence:

```
@param x  the x-coordinate. Measured in pixels.
```

- If you prefer starting with a sentence, capitalize it and end it with a period:

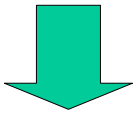
```
@param x  Specifies the x-coordinate, measured in pixels.
```

- When writing multiple sentences, follow normal sentence rules:

```
@param x  Specifies the x-coordinate. Measured in pixels.
```

- **@return** *description*
  - Adds a "Returns" section with the *description* text. This text should describe the return type and permissible range of values. This tag is valid only in a doc comment for a method.
  - The @return tag is required for every method that returns something other than void, even if it is redundant with the method description.

- **@see** *reference*
  - Adds a "See Also" heading with a link or text entry that points to *reference*.
  - A doc comment may contain any number of @see tags, which are all grouped under the same heading.
  - This tag is valid in any doc comment: overview, package, class, interface, constructor, method or field.
  - This tag has 3 forms



- **@see** "*string*"

Adds a text entry for *string*. No link is generated.

The Javadoc tool distinguishes this from the previous

cases by looking for a double-quote (") as the first

character. For example:

```
@see "The Java Programming Language"
```

See Also:
"The Java Programming Language"

- **@see** *label*
  - Adds a link as defined by *URL#value*. The *URL#value* is a relative or absolute URL. For example:

@see Java Spec

See Also:
Java Spec

- **@see** *package.class#member label*
  - Adds a link, with visible text *label*, that points to the documentation for the specified name in the Java Language that is referenced. The *label* is optional; if omitted, *package.class.member* will appear, suitably shortened relative to the current class and package
  - <code> is always included around the visible text
  - ***package.class#member*** is any valid program element name that is referenced -- a package, class, interface, constructor, method or field name -- except that the character ahead of the member name should be a hash character (#).
    - » The *class* represents any top-level or nested class or interface.
    - » The *member* represents any constructor, method or field (not a nested class or interface).
    - » If this name is in the documented classes, the Javadoc tool will automatically create a link to it.
    - » To create links to external referenced classes, use the -link option.
  - ***label*** is optional text that is visible as the link's label.
    - The *label* can contain whitespace.
    - A space is the delimiter between *package.class#member* and *label*. A space inside parentheses does not indicate the start of a label, so spaces may be used between parameters in a method.

@see example

```
/**
* @see String#equals(Object) equals
*/
```

```
<dl>
<dt><b>See Also:</b>
<dd><a
  href="../../java/lang/String#equals(java.lang.Object)"><code>equals<co
  de></a>
 </dl>
```

Specifying a name

- This *package.class#member* name can be either fully-qualified, such as java.lang.String#toUpperCase() or not, such as String#toUpperCase() or #toUpperCase(). If less than fully-qualified, the Javadoc tool uses the normal Java compiler search order to find it

Referencing a member of the current class
@see *#field*
@see *#method(Type, Type,...)*
@see *#method(Type argname, Type argname,...)*
@see *#constructor(Type, Type,...)*
@see *#constructor(Type argname, Type argname,...)*

Referencing another class in the current or imported packages
@see *Class#field*
@see *Class#method(Type, Type,...)*
@see *Class#method(Type argname, Type argname,...)*
@see *Class#constructor(Type, Type,...)*
@see *Class#constructor(Type argname, Type argname,...)*
@see *Class.NestedClass*
@see *Class*

Referencing an element in another package (fully qualified)
@see *package.Class#field*
@see *package.Class#method(Type, Type,...)*
@see *package.Class#method(Type argname, Type argname,...)*
@see *package.Class#constructor(Type, Type,...)*
@see *package.Class#constructor(Type argname, Type argname,...)*
@see *package.Class.NestedClass*
@see *package.Class*
@see *package*

Search order for @see

- the Javadoc tool will process a @see tag that appears in a source file (.java), package file (package.html) or overview file (overview.html). In the latter two files, you must fully-qualify the name you supply with @see.
- When the Javadoc tool encounters a @see tag in a .java file that is *not* fully qualified, it searches for the specified name in in this order:
    1. the current class or interface
    2. any enclosing classes and interfaces, searching closest first
    3. any superclasses and superinterfaces, searching closest first
    4. the current package
    5. any imported packages, classes and interfaces, searching in the order of the import statement

    ---after it searches through the current class and its enclosing class E, it will search through E's superclasses before E's enclosing classes.

- **@serial** *field-description* | include | exclude
- Used in the doc comment for a default serializable field. An optional *field-description* should explain the meaning of the field and list the acceptable values. If needed, the description can span multiple lines. The standard doclet adds this information to the serialized form page.
- If a serializable field was added to a class some time after the class was made serializable, a statement should be added to its main description to identify at which version it was added.
- The include and exclude arguments identify whether a class or package should be included or excluded from the serialized form page. They work as follows:

- A public or protected class that implements Serializable is *included* unless that class (or its package) is marked @serial exclude.
- A private or package-private class that implements Serializable is *excluded* unless that class (or its package) is marked @serial include.

- **@serialField** *field-name field-type field-description*
  - Documents an ObjectStreamField component of a Serializable class's serialPersistentFields member. One @serialField tag should be used for each ObjectStreamField component.
- **@serialData** *data-description*
  - The *data-description* documents the types and order of data in the serialized form. Specifically, this data includes the optional data written by the writeObject method and all data (including base classes) written by the Externalizable.writeExternal method.
  - The @serialData tag can be used in the doc comment for the writeObject, readObject, writeExternal, and readExternal methods.

- **@since** *since-text*
  - Adds a "Since" heading with the specified *since-text* to the generated documentation. The text has no special internal structure.
  - This tag is valid in any doc comment: overview, package, class, interface, constructor, method or field.
  - This tag means that this change or feature has existed since the software release specified by the *since-text*. For example:

    @since 1.4

    - For source code in the Java platform, this tag indicates the version of the Java platform API specification (not necessarily when it was added to the reference implementation).
    - Multiple @since tags are allowed and are treated like multiple @author tags.
    - You could use multiple tags if the program element is used by more than one API.

- When a package is introduced, specify an @since tag in its package description and each of its classes.
- When a class (or interface) is introduced, specify one @since tag in its class description and no @since tags in the members. Add an @since tag only to members added in a later version than the class.
- If a member changes from protected to public in a later release, the @since tag would not change.

- **@throws** *class-name description*
  - The @throws and @exception tags are synonyms.
  - Adds a "Throws" subheading to the generated documentation, with the *class-name* and *description* text.
  - The *class-name* is the name of the exception that may be thrown by the method.
  - This tag is valid only in the doc comment for a method or constructor.
  - If this class is not fully-specified, the Javadoc tool uses the search order to look up this class.
  - Multiple @throws tags can be used in a given doc comment for the same or different exceptions.
  - To ensure that all checked exceptions are documented, if a @throws tag does not exist for an exception in the throws clause, the Javadoc tool automatically adds that exception to the HTML output (with no description) as if it were documented with @throws tag
    - The @throws documentation is copied from an overridden method to a subclass only when the exception is explicitly declared in the overridden method.

@throws example

- A @throws tag should be included for any checked exceptions (declared in the throws clause), and for any unchecked exceptions that the caller might reasonably want to catch, with the exception of NullPointerException.
- Errors should not be documented as they are unpredictable.

```
/**
* @throws IOException If an input or output exception occurred
*/
public void f() throws IOException { // body }
```

@throws example 2

- a method that takes an index and uses an array internally should *not* be documented to throw an ArrayIndexOutOfBoundsException, as another implementation could use a data structure other than an array internally.
- It is, however, generally appropriate to document that such a method throws an IndexOutOfBoundsException.

- **{@value}**
  - When used in the doc comment of a static field, displays the value of the constant. These are the values displayed on the Constant Field Values page. This tag is valid only in doc comments for fields.

- **@version** *version-text*
  - Adds a "Version" subheading with the specified *version-text* to the generated docs when the -version option is used.
  - This tag is intended to hold the current version number of the software that this code is part of (as opposed to @since, which holds the version number where this code was introduced).
  - The *version-text* has no special internal structure.
  - A doc comment may contain multiple @version tags.
  - If it makes sense, you can specify one version number per @version tag
    - » the Javadoc tool inserts a comma (,) and space between names.
  - or multiple version numbers per tag.
    - » the entire text is simply copied to the generated document without being parsed. Therefore, you can use multiple names per line if you want a localized name separator other than comma.

Overview Tags	
@see @since @author @version {@link} {@linkplain} {@docRoot}	appear in the documentation comment for the overview page (overview.html).

Package Tags	
@see @since @serial @author @version {@link} {@linkplain} {@docRoot}	can appear in the documentation comment for a package (package.html). The @serial tag can only be used here with the include or exclude argument.

Class/Interface Tags	
@see @since @deprecated @serial @author @version {@link} {@linkplain} {@docRoot}	appear in the documentation comment for a class or interface. The @serial tag can only be used here with the include or exclude argument.

Class comment example

```
/**
*A class representing a window on the screen.
* For example:
* <pre>
* Window win = new Window(parent);
* win.show();
* </pre>
*
* @author Sami Shaio
* @version %I%, %G%
* @see java.awt.BaseWindow
* @see java.awt.Button
*/
class Window extends BaseWindow { ... }
```

Field Tags

@see
@since

@deprecated
@serial
@serialField
{@link}

{@linkplain}

{@docRoot}

{@value}

An example of a field comment:

```
/**
*The X-coordinate of the component.
*
* @see #getLocation()
*/
int x = 1263732;
```

Method/Constructor Tags

@see
@since

@deprecated
@param
@return
@throws
and @exception
@serialData
{@link}

{@linkplain}

{@inheritDoc}

{@docRoot}

except for @return, which cannot appear in a constructor.

{@inheritDoc}, which has certain restrictions.

The @serialData tag can only be used in the doc comment for certain serialization methods.

```
/**
*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 StringIndexOutOfRangeException
* 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) { ... }
```

Guidelines

- **Implementation-Independence**
  - Define clearly what is required and what is allowed to vary across platforms/implementations.
  - If you must document implementation-specific behavior, please document it in a separate paragraph.
  - If the implementation varies according to platform, then specify "On <platform>" at the start of the paragraph.
  - In other cases that might vary with implementations on a platform you might use the lead-in phrase "Implementation-Specific. For example:
    - On Windows systems, ……

- **Use <code> style for keywords and names.** Keywords and names are offset by <code>...</code> when mentioned in a description. This includes:
  - Java keywords
  - package names
  - class names
  - method names
  - interface names
  - field names
  - argument names
  - code examples

- **Use in-line links economically**
  - adding a link:
    - Only if the user might actually want to click on it for more information (in your judgment), and
    - Only for the first occurrence of each API name in the doc comment (don't bother repeating a link)
    - It is not necessary to link to API in the java.lang package, it is well-known anyway.

- **Omit parentheses for the general form of methods and constructors**
  - add(Object) and add(int, Object)
  - if referring to both forms of the method, omit the parentheses altogether.
- **Okay to use phrases instead of complete sentences.**
- **Use 3rd person (descriptive) not 2nd person (prescriptive).**
  - The description is in 3rd person declarative rather than 2nd person imperative.
    - Gets the label. (preferred)
    - Get the label. (avoid)

- **Method descriptions begin with a verb phrase.** For example:
  - "Gets the label of this button."
- **Class/interface/field descriptions can omit the subject and simply state the object.**
  - "A button label."
- **Use "this" instead of "the" when referring to an object created from the current class.**
  - "Gets the toolkit for this component."

```
/**
* Sets the tool tip text.
*
* @param text the text of the tool tip
*/
public void setToolTipText(String text) {
```

Unnecessary comment must be avoided

```
/**
* Registers the text to display in a tool tip. The text
* displays when the cursor lingers over the component.
*
* @param text the string to display. If the text is null,
* the tool tip is turned off for this component.
*/
public void setToolTipText(String text) {
```

Good and useful comment

- **Be clear when using the term "field".** Be aware that the word "field" has two meanings:
  - static field, which is another term for "class variable"
  - text field, as in the TextField class.
- **Avoid Latin**
  - use "also known as" instead of "aka",
  - use "that is" or "to be specific" instead of "i.e.",
  - use "for example" instead of "e.g.", and
  - use "in other words" or "namely" instead of "viz."

Order of Tags

- Include tags in the following order:
  - @author (classes and interfaces only, required)
  - @version (classes and interfaces only, required) (see footnote 1)
  - @param (methods and constructors only)
  - @return (methods only)
  - @exception (@throws is a synonym added in Javadoc 1.2)
  - @see
  - @since
  - @serial (or @serialField or @serialData)
  - @deprecated

Ordering Multiple Tags

- Multiple @author tags should be listed in chronological order, with the creator of the class listed at the top.
- Multiple @param tags should be listed in argument-declaration order.
- Multiple @throws tags (also known as @exception) should be listed alphabetically by the exception names.

Ordering Multiple @see tags

- from nearest to farthest access, from least-qualified to fully-qualified
- methods and constructors are in "telescoping" order, which means the "no arg" form first, then the "1 arg" form, then the "2 arg" form.

- @see #field
- @see #Constructor(Type, Type...)
- @see #Constructor(Type id, Type id...)
- @see #method(Type, Type,...)
- @see #method(Type id, Type, id...)
- @see Class
- @see Class#field
- @see Class#Constructor(Type, Type...)
- @see Class#Constructor(Type id, Type id)
- @see Class#method(Type, Type,...)
- @see Class#method(Type id, Type id,...)
- @see package.Class
- @see package.Class#field
- @see package.Class#Constructor(Type, Type...)
- @see package.Class#Constructor(Type id, Type id)
- @see package.Class#method(Type, Type,...)
- @see package.Class#method(Type id, Type, id)
- @see package

Documenting Anonymous Inner Classes

- do it in a doc comment of its outer class, or another closely associated class.
- For example, anonymous inner class TreeSelectionListener in a method makeTree that returns a JTree object

```
/**
* The method used for creating the tree. Any structural modifications
* to the display of the Jtree should be done by overriding this method.
* This method adds an anonymous TreeSelectionListener to the returned JTree.
* Upon receiving TreeSelectionEvents, this listener calls refresh with
* the selected node as a parameter.
*/
public JTree makeTree(AreaInfo ai){ }
```