Checklist: How to write javadoc

subtitle

Author:
a-t-0

10-08-2019

1 Introduction

This document is based on source: `https://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#exampleresult` and contains:

1. list of tips/basics on how to write javadoc in section 3.
2. Checklist for class documentation.
3. Checklist for method documentation.
4. Checklist for etiquette/requirements the code in general derived from javadoc.

2 TODO

1. Distil the sentences for the checklist per Javadoc type:
   (a) class
   (b) constructor
   (c) method
   (d) field
2. Convert the information of those sentences into checklist format.
3. Determine and document how to "automatically" "compile" the HTML/API/booklet format of the Javadoc.

3 Javadoc basics

Javadoc is not only read by someone that opens your `.java` files/project in a text editor and/or IDE, but there is some option that reads the javadoc from some pre-defined format into a different/website format. There are two types of writing Javadoc:

- **API Documentation (API docs) is for developers/programmers.** Writing it for testers, that test edgecases for cross platform compatability. This requires writing all edgecases, expected/intendend behaviors and platform dependent information.
- **API Specifications (API specs) is for testers/cross platform stuffs** Writing it for fellow developers so that they can build upon your work.

3.1 General Javadoc info

The listed oracle documentation titled:`How to Write Doc Comments for the Javadoc Tool` is actually designed for the former. So how to write the gritty details of methods etc, instead of writing it for developers. Nevertheless, the info relevant for writing developer javadoc is distilled from it.

1. Javadoc is written in HTML. A nice overview of the text formatting options built into HTML is given here: `https://www.simplehtmlguide.com/cheatsheet.php`
2. The Javadoc generates the Java API Specification. This Java API Specification is a guide/explanation on how someone should interact with your code. (So not an API for general java .jar files or some sort.).
3. They prefer to write that a bit more formal/different as:
   `The Java Platform API Specification is a contract between callers and implementations.`

4 Concrete javadoc writing tips

Assuming you use Eclipse (=opensource), you can also use paid versions as: IntelliJ Pycharm, Spyder or whatever, notepad to work in java projects.:

1. You write javadocs above each:
   (a) class
   (b) method
   (c) constructor
   (d) field

4.1 First sentence

1. The first compact sentence (till dot) of every Javadoc is a short summary of the method, as Javadoc automatically places it in the method summary table (and index).
   (a) You can escape a dot in the first line that should not be the end of the sentence, for example:
   - This is a simulation of Prof. Knuth's MIX computer.
   - (The summary would be: "This is a simulation of Prof." which is not enough.)

   with either:
   - `This is a simulation of Prof.  Knuth's MIX computer.`

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

2. If it's an overloaded method/constructor, make a difference in the first line that explains the difference between the two. For example (ps. foo is the word used to indicate a random method named foo, or whatever.):
   (a)
```
/**
* Class constructor.
*/
foo() {
 ...
```
   (b)
```
/**
* Class constructor specifying number of objects to create.
*/
foo(int n) {
 ...
```

4.2 General text/typing

1. Use 3rd person (descriptive) not 2nd person (prescriptive).
   (a) 2nd person: `You can wait in here and make yourself at home`
   (b) 3rd person: `Tiffany used him/her prize money from the science fair to buy her-/himself a new micros`
2. The description is in 3rd person declarative rather than 2nd person imperative.
   (a) Gets the label. (preferred)
   (b) Get the label. (avoid)
3. Method descriptions begin with a verb phrase.
   (a) `Gets the label of this button. (preferred)`
   (b) This method gets the label of this button. (avoid)
4. Class/interface/field descriptions can omit the subject and simply state the object. These API often describe things rather than actions or behaviors E.g.:
   (a) A button label. (preferred)
   (b) This field is a button label. (avoid)
5. Use "this" instead of "the" when referring to an object created from the current class. For example, the description of the getToolkit method should read as follows:

(a) Gets the toolkit for this component. (preferred)

(b) Gets the toolkit for the component. (avoid)

6. If a method is named "findMaxHeight()" write:

   (a) Registers the height in inches with a minimum of 80 feet.

   (b) Finds the maximum height. (avoid)(because it doesn't add anything new/valuable):

7. Don't confuse (static) field (class variable) with the `TextField` class. Be clear which one you use by distinguishing

   (a) `static field` (appearently those class fields are always static even if they're of a non-static object).

   (b) `text field` E.g. can contain date field or number field etc.

8. Don't use abbreviations or latin.

   (a) For example, to be specific, namely (preferred)

   (b) E.g., i.e., viz (avoid)

4.3 Tags

(a) The following tags are (always) required when method takes args or returns something:

   i. @Param (Must write something even the description is obvious)

   ii. @Return if it is redundant with method description. This is to help searching your code.)

Bonus gem:

- Check if you can state something more specific for the tag comment. (e.g. limit of int value if it is custom to that method).

(b) The order of tags is:

   i. @author (classes and interfaces only, required)

   ii. @version (classes and interfaces only, required. See footnote 1)

   iii. @param (methods and constructors only)

   iv. @return (methods only)

   v. @exception (@throws is a synonym added in Javadoc 1.2)

   vi. @see

   vii. @since

   viii. @serial (or @serialField or @serialData)

   ix. @deprecated (see How and When To Deprecate APIs)

(c) if you have multiple tags in a single class:

- @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. This makes it easier to visually match the list to the declaration.
- Multiple @throws tags (also known as @exception) should be listed alphabetically by the exception names.
- Multiple @see tags should be ordered as follows, which is roughly the same order as their arguments are searched for by javadoc, basically from nearest to farthest access, from least-qualified to fully-qualified, The following list shows this progression. Notice the methods and constructors are in "telescoping" order, which means the "no arg" form first, then the "1 arg" form, then the "2 arg" form, and so forth. Where a second sorting key is needed, they could be listed either alphabetically or grouped logically.

   i. `@see #field`

   ii. `@see #Constructor(Type, Type...)`

   iii. `@see #Constructor(Type id, Type id...)`

   iv. `@see #method(Type, Type,...)`

   v. `@see #method(Type id, Type, id...)`

   vi. `@see Class`

   vii. `@see Class#field`

   viii. `@see Class#Constructor(Type, Type...)`

   ix. `@see Class#Constructor(Type id, Type id)`

x. `@see Class#method(Type, Type,...)`
xi. `@see Class#method(Type id, Type id,...)`
xii. `@see package.Class`
xiii. `@see package.Class#field`
xiv. `@see package.Class#Constructor(Type, Type...)`
xv. `@see package.Class#Constructor(Type id, Type id)`
xvi. `@see package.Class#method(Type, Type,...)`
xvii. `@see package.Class#method(Type id, Type, id)`
xviii. `@see package`

4.4 HTML typing

1. Use `<code>..</code>` when you refer to:
   (a) Java keywords (E.g. "if","for", etc. `https://en.wikipedia.org/wiki/List_of_Java_keywords`.
   (b) package names
   (c) class names
   (d) method names
   (e) interface names
   (f) field names
   (g) argument names
   (h) code examples
2. You can refer to links in javadoc with: `.. which can be found in {@link URL}...`
3. You can separate paragraphs, if needed, with: `<p>`

4.5 Implementation independence

List the dependencies this method implies. This facilitates principle:"write once, run anywhere."

4.6 Automated checking of Javadoc

Oracle created DocCheck which is a plugin that automatically lists the errors in your Javadoc.

4.7 Methods

1. Source:`https://www.geeksforgeeks.org/methods-in-java/`
2. Method name is: verb starting with lower letter (can be followed by adjective or noun). E.g.

```
findSum
computeMax
setX
getX
```

4.8 Overloading

1. If you overload a method, e.g. `add(int number) add(int number,String word)` only use the types when you to ONE specific method of the two. E.g `* as can be seen in method <code>add(int,String)</code> but not i`
2. If you refer to both methods, just use: `as can be seen in both methods <code>add</code>`

1.

2.