Version:10-01-2017

Author: Alexander Boffin

**Java Coding Standards**

Why is it necessary to come to a consensus about how to write code? Everybody has his own style of programming but 80% of the lifetime cost of software comes from maintenance. This maintenance work (updating, debugging….) isn’t often done by the author of the code. Because of this it is necessary to write in an understandable and clear way. If this isn’t done its more difficult for those that need to keep your programs up and running. There are several elements that can improve the readability of your code.

Inhoudsopgave

[A) The Comments: 1](#_Toc471808719)

[B) Line Length: 2](#_Toc471808720)

[C) Declarations 2](#_Toc471808721)

[D) Using Blank Lines: 2](#_Toc471808722)

[E) Naming Conventions: 3](#_Toc471808723)

[F) Arrays 4](#_Toc471808724)

[G) Parenthesis 4](#_Toc471808725)

# A) The Comments:

There are several ways to add comments

1. **//**everything behind the two backslashes won’t be read when executing the file.
2. **/\***Comments can be placed here**\*/**

&

**/\***-------------------can be used to make Heathers---------------------------**\*/**

1. /\* Like this

\* comments can be placed over multiple

**\***lines**\*/**

&

*/\* ----------------------------------------------------------------  
Heather  
\*------------------------------------------------------------------\*/*

1. /\*\*This will create a Javadoc, what can be called upon

\*More descriptive as 3, they are meant as reference documents

\*Will say what the function does, behaves, …

\*/

(!) If using 1. or 2. after a statement it is recommended to use a tab of 4 or 8 whitespaces. This makes the comment more visible for the reviewer.

# B) Line Length:

Lines longer than 70 characters should be avoided. Wrapping up a line is preferably done;

* After a “,”
* Before an operator “.”
* By aligning a new line with the previous line. Example:

count = number.calculate(bytes, offset, length,

value, 0, estimatedCount);

Avoid files longer than 2000 lines=>try to break these into several classes that then become integrated.

C) Declarations

=>specify the identifier and type of elements (like int a;)

It is recommended to use only 1 declaration for each line. This helps to keep an overview of all the elements.

When initiating them do this where they are needed in the file and preferably at the beginning of blocks (like direct after the “{“ in loops).

# D) Using Blank Lines:

Blank lines can help to make the structure more readable by separating it into separate fractions. This is more appealing for reading the code and helps find the necessary blocks faster.

Use 2 lines

* Between sections of a source file
* Between class and interface definitions

Use 1 line:

* Between methods
* Between local variables and the first statement
* Before a single line-statement
* Between logical sections inside the method to improve readability
* Before/after a comment

# E) Naming Conventions:

When starting to code you can rapidly use words that are for you very clear. But this isn’t the same for other people. Admitted that using funny names can be very hard to resist. But imagine your colleagues reading the code and don’t find the logic in the code because of it, leading to delays and errors in the final product. There are several elements you can do on this point:

Generally:

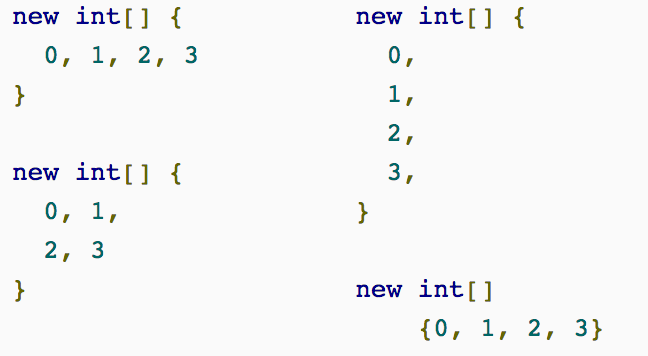
* Use English descriptors that give an indication of the elements purpose
* Use terminology that is applicable for the field you are writing the program for. (Using the name customer and not client for example in the businessworld)
* Don’t use abbreviations. If used make sure to document it properly/visibly (like in a comment) and use it sparingly!
* Avoid using similar or very long names

More specific

* Packages: written in lowercase lettering
  + Example: pakage exercisesselenium
* Classes:
  + Use nouns
  + Capitalise each first letter of the internal words
  + Keep short and descriptive
  + Use whole words
  + Example: class ImageSpirit;
* Interfaces: same as classes
* Methods:
  + Use verbs
  + First letter lowercase, subsequent first letter of internal word uppercase.
  + Specific prefixes in some cases:
    - “get” 🡪 if method returns an element
    - “is”🡪if it returns a boolean variable
    - “set”🡪if the method is mutating an element that is given to it.
  + Avoid using cascading names for variables given to the method. For example methodExample(string1, string2) 🡪 methodExample(tooperateon, additor)
  + Example: runFaster();
* Variables:
  + Lowercase lettering
  + Short & meaningful (is intended to do what?)
  + Don’t use one-character names. Except with “throwaway” variables (I, j, k, m & n is used for integers and c, d & e for characters)
  + Example: int mywidth
* Constants=>should be avoided as much as possible
  + Uppercase lettering with “\_” between internal words
  + Example: MIN\_WIDTH=4;

# F) Arrays

Arrays can be written in several ways. Some of these can make the code more meaningful to others. In the figure below this is illustrated in a simple case. Look what gives the better overview depending on the situation at hand.



# G) Parenthesis

It might sound silly but parentheses can really help improve the readability of the code. What is more readable/understandable:

Int number= 1+6+7\*4+9\*2+7+3

or

Int number= 1+6+(7\*4)+(9\*2)+7+3

Both will give the same outcome, but the second is easier to read. Several elements can be made more readable in your code if you use the parentheses. Try to use them as much as possible (if applicable and it makes the code more readable).