**Version History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. No.** | **Authors** | **Date** | **Reviewers** | **Review Date** | **Release Date** |
| 1.0 | Application Development Team | 27-Aug-2018 | QMF | 31-Aug-2018 | 03-Sep-2018 |
| 2.0 | Application Development Team | 10-Dec-2019 | QMF | 13-Dec-2019 | 16-Dec-2019 |
| 3.0 | Application Development Team | 02-Nov-2020 | QMF | 06-Nov-2020 | 10-Nov-2020 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Change History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ver. No.** | **Section** | **Date** | **Change Information** | **RFC No.** |
| 1.0 | All | 03-Sep-2018 | New Release | - |
| 2.0 | All | 16-Dec-2019 | Annual Review | - |
| 3.0 | All | 02-Nov-2020 | Annual Review | - |
|  |  |  |  |  |
|  |  |  |  |  |

# STANDARDS FOR CODING IN JAVA

1. **Objective**

The objective of this document is to define the coding standards to be followed while developing application using ‘Java’.

1. **Scope**

This standard applies to all products / projects developed using Java.

1. **References to (checklists, forms, guidelines, lists, standards, Templates, other processes)**

| **Item** | **Description** | **ID** |
| --- | --- | --- |
| **Checklists** | -- | -- |

1. **Document Organization**

This document is organized as follows:

* Beginning Comments
* Package and Import statements
* Class and interface declarations

## Beginning Comments

* + - All source files should begin with a c-style comment that lists the class name, version information, date, and copyright notice:

/\*

\* Classname

\*

\* Version information

\*

\* Date

\*

\* Copyright notice

\*/

## Package and Import Statements

* + - The first non-comment line of most Java source files is a package statement. After that, import statements can follow
    - For example: package java.awt; import java.awt.peer.CanvasPeer;
    - Note: The first component of a unique package name is to be written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries

## Class and Interface Declarations

| # | Part of Class / Interface Declaration | Notes |
| --- | --- | --- |
| 1 | Class/interface documentation comment (/\*\*...\*/) | See Section 5.4 for information on what should be in this comment. |
| 2 | Class or interface statement |  |
| 3 | Class/interface implementation comment (/\*...\*/), if necessary | This comment should contain any class-wide or interface-wide information that wasn't appropriate for the class/interface documentation comment. |
| 4 | Class (static) variables | First the public class variables, then the protected, then package level (no access modifier), and then the private. |
| 5 | Instance variables | First public, then protected, then package level (no access modifier), and then private. |
| 6 | Constructors |  |
| 7 | Methods | These methods should be grouped by functionality rather than by scope or accessibility. For example, a private class method can be in between two public instance methods. The goal is to make reading and understanding the code easier. |

1. **Indentation**

* Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Tabs must be set exactly every 8 spaces

## Line Length

* + - Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.
    - Note: Examples for use in documentation should have a shorter line length-generally no more than 70 characters.

## Wrapping Lines

* + - When an expression will not fit on a single line, break it according to these general principles:
  + Break after a comma
  + Break before an operator
  + Prefer higher-level breaks to lower-level breaks
    - Align the new line with the beginning of the expression at the same level on the previous line
    - If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead
    - Here are some examples of breaking method calls:

someMethod (longExpression1, longExpression2, longExpression3,

longExpression4, longExpression5);

var = someMethod1(longExpression1,

someMethod2(longExpression2,

longExpression3));

Following are two examples of breaking an arithmetic expression. The first is preferred, since the break occurs outside the parenthesized expression, which is at a higher level.

longName1 = longName2 \* (longName3 + longName4 - longName5)

+ 4 \* longname6; // PREFER

longName1 = longName2 \* (longName3 + longName4

- longName5) + 4 \* longname6; // AVOID

Following are two examples of indenting method declarations. The first is the conventional case. The second would shift the second and third lines to the far right if it used conventional indentation, so instead it indents only 8 spaces.

//CONVENTIONAL INDENTATION

someMethod(int anArg, Object anotherArg, String yetAnotherArg,

Object andStillAnother) {

...

}

//INDENT 8 SPACES TO AVOID VERY DEEP INDENTS

private static synchronized horkingLongMethodName(int anArg,

Object anotherArg, String yetAnotherArg,

Object andStillAnother) {

...

}

* When an expression will not fit on a single line, break it according to these general principles:
  + Break after a comma.
  + Break before an operator.
  + Prefer higher-level breaks to lower-level breaks.
* Align the new line with the beginning of the expression at the same level on the previous line.

If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Here are some examples of breaking method calls:

Line wrapping for if statements should generally use the 8-space rule, since conventional (4 space) indentation makes seeing the body difficult. For example:

//DON'T USE THIS INDENTATION

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6)) { //BAD WRAPS

doSomethingAboutIt(); //MAKE THIS LINE EASY TO MISS

}

//USE THIS INDENTATION INSTEAD

if ((condition1 && condition2)

|| (condition3 && condition4)

||!(condition5 && condition6)) {

doSomethingAboutIt();

}

//OR USE THIS

if ((condition1 && condition2) || (condition3 && condition4)

||!(condition5 && condition6)) {

doSomethingAboutIt();

}

Here are three acceptable ways to format ternary expressions:

alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta

: gamma;

alpha = (aLongBooleanExpression)

? beta

: gamma;

1. **Comments** 
   * + Implementation comments are meant for commenting out code or for comments about the particular implementation
     + Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain information that is relevant to reading and understanding the program
     + Avoid duplicating information that is present in (and clear from) the code
     + Avoid any comments that are likely to get out of date as the code evolves
     + Comments should never include special characters such as form-feed and backspace

## Implementation Comment Formats

* + - Programs can have four styles of implementation comments: block, single-line, trailing, and end-of-line

## Block Comments

* + - Block comments are used to provide descriptions of files, methods, data structures and algorithms. Block comments should be used at the beginning of each file and before each method
    - Block comments inside a function or method should be indented to the same level as the code they describe
    - A block comment should be preceded by a blank line to set it apart from the rest of the code

/\*

\* Here is a block comment.

\*/

Block comments can start with /\*-, which is recognized by **indent**(1) as the beginning of a block comment that should not be reformatted. Example:

/\*-

\* Here is a block comment with some very special

\* formatting that I want indent(1) to ignore.

\*

\* one

\* two

\* three

\*/

## Single-Line Comments

* + - These comment should be single line indented to the level of the code that follows
    - A single-line comment should be preceded by a blank line

/\* Handle the condition. \*/

....

## Trailing Comments

* + - Very short comments can appear on the same line as the code they describe, but should be shifted far enough to separate them from the statements
    - If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting
    - E.g.

if (a == 2) {

return TRUE; /\* special case \*/

} else {

return isPrime(a); /\* works only for odd a \*/

}

## End-Of-Line Comments

* + - The // comment delimiter can comment out a complete line or only a partial line. It shouldn't be used on consecutive multiple lines for text comments; however, it can be used in consecutive multiple lines for commenting out sections of code.
    - Examples of all three styles follow:

if (foo > 1) {

// Do a double-flip.

...

}

else {

return false; // Explain why here.

}

//if (bar > 1) {

//

// // Do a triple-flip.

// ...

//}

//else {

// return false;

//}

1. **Declaration**

## Number Per Line

* + - Should give only one declaration per line since it encourages commenting
    - int level; // indentation level
    - int size; // size of table
    - is preferred over

## Initialization

* + - Initialize local variables where they're declared
    - The only reason not to initialize a variable where it's declared is if the initial value depends on some computation occurring first

## Placement

* + - Declarations should be done only at the beginning of blocks. (A block is any code surrounded by curly braces "{" and "}".)

void myMethod() {

int int1 = 0; // beginning of method block

if (condition) {

int int2 = 0; // beginning of "if" block

...

}

}

* + - The one exception to the rule is indexes of for loops, which in Java can be declared in the for statement:

int count;

...

myMethod() {

if (condition) {

int count = 0; // AVOID!

...

}

...

}

## Class and Interface Declarations

* + - When coding Java classes and interfaces, the following formatting rules should be followed:
  + No space between a method name and the parenthesis "(" starting its parameter list
  + Open brace "{" appears at the end of the same line as the declaration statement
  + Closing brace "}" starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the "}" should appear immediately after the "{"

class Sample extends Object {

int ivar1;

int ivar2;

Sample(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int emptyMethod() {}

...

}

* + Separated methods by a blank line

1. **Statements**

## Simple statements

* + - Each line should contain at most one statement
    - Example:

argv++; // Correct

argc--; // Correct

argv++; argc--; // AVOID!

## Compound Statements

* + - Compound statements are statements that contain lists of statements enclosed in braces "{statements}"
    - Examples:
  + The enclosed statements should be indented one more level than the compound statement
  + The opening brace should be at the end of the line that begins the compound statement; the closing brace should begin a line and be indented to the beginning of the compound statement
  + Braces are used around all statements, even single statements, when they are part of a control structure, such as a if-else or for statement. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces

## Return Statements

* + - A return statement with a value should not use parentheses unless they make the return value more obvious in some way
    - Example:

return;

return myDisk.size();

return (size ? size : defaultSize);

## If, if-else, if else-if else Statements

* + - The if-else class of statements should have the following form:

if (*condition*) {

*statements*;

}

if (*condition*) {

*statements*;

} else {

*statements*;

}

if (*condition*) {

*statements*;

} else if (*condition*) {

*statements*;

} else {

*statements*;

}

## For Statements

* + - A for statement should have the following form

|  |
| --- |
| for (initialization; condition; update) { |
| statements; |
| } |

* + - An empty for statement (one in which all the work is done in the initialization, condition, and update clauses) should have the following form: for (initialization; condition; update);
    - When using the comma operator in the initialization or update clause of a for statement, avoid the complexity of using more than three variables. If needed, use separate statements before the for loop (for the initialization clause) or at the end of the loop (for the update clause)

## While Statements

* + - A while statement should have the following form:

|  |
| --- |
| while (condition) { |
| statements; |
| } |

* + - An empty while statement should have the following form: while (condition);

## Do-While Statements

* + - A do-while statement should have the following form:

|  |
| --- |
| do { |
| statements; |
| } while (condition); |

## Switch Statements

* + - A switch statement should have the following form:

|  |
| --- |
| switch (condition) { |
| case ABC: |
| statements; |
| /\* falls through \*/ |
|  |
| case DEF: |
| statements; |
| break; |
|  |
| case XYZ: |
| statements; |
| break; |
|  |
| default: |
| statements; |
| break; |
| } |

* + - Every time a case falls through (doesn't include a break statement), add a comment where the break statement would normally be. This is shown in the preceding code example with the /\* falls through \*/ comment
    - Every switch statement should include a default case. The break in the default case is redundant, but it prevents a fall-through error if later another case is added

## Try-Catch Statements

* + - A try-catch statement should have the following format:

|  |
| --- |
| try { |
| statements; |
| } catch (ExceptionClass e) { |
| statements; |
| } |

* + - A try-catch statement may also be followed by finally, which executes regardless of whether or not the try block has completed successfully

|  |
| --- |
| try { |
| statements; |
| } catch (ExceptionClass e) { |
| statements; |
| } finally { |
| statements; |
| } |

1. **Naming Conventions**

| **Identifier Type** | **Rules for Naming** | **Examples** |
| --- | --- | --- |
| Packages | unique package name is to be written in all-lowercase ASCII letters and should be one of the top-level domain names, like com, edu, gov, mil, net, org, | com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese |
| Classes | Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Use whole words-avoid acronyms and abbreviations | class Raster; class ImageSprite; |
| Interfaces | Interface names should be capitalized like class names. | interface RasterDelegate; interface Storing; |
| Methods | Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. | run(); runFast(); getBackground(); |
| Variables | * Variable names should not start with underscore \_ or dollar sign $ characters * Variable names should be short * The choice of a variable name should be mnemonic * One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters | int i;  char c;  float myWidth; |
| Constants | The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("\_") | static final int MIN\_WIDTH = 4;  static final int MAX\_WIDTH = 999;  static final int GET\_THE\_CPU = 1; |

1. **Coding Practices**

## Providing Access to Instance and Class Variables

* + - Don't make any instance or class variable public without good reason. Often, instance variables don't need to be explicitly set or gotten-often that happens as a side effect of method calls
    - One example of appropriate public instance variables is the case where the class is essentially a data structure, with no behavior. In other words, if you would have used a struct instead of a class (if Java supported struct), then it's appropriate to make the class's instance variables public

## Referring to Class Variables and Methods

* + - Avoid using an object to access a class (static) variable or method. Use a class name instead
    - For example:

|  |
| --- |
| classMethod(); //OK |
| AClass.classMethod(); //OK |
| anObject.classMethod(); //AVOID |

## Constants

* + - Numerical constants (literals) should not be coded directly, except for -1, 0, and 1, which can appear in a for loop as counter values

## Variable Assignments

* + - Avoid assigning several variables to the same value in a single statement. It is hard to read
    - Example:

|  |
| --- |
| fooBar.fChar = barFoo.lchar = 'c'; // AVOID! |
| Do not use the assignment operator in a place where it can be easily confused with the equality operator. Example: |
| if (c++ = d++) { // AVOID! (Java disallows) |
| ... |
| } |
| should be written as |
| if ((c++ = d++) != 0) { |
| ... |
| } |

* + - Do not use embedded assignments in an attempt to improve run-time performance. This is the job of the compiler
    - Example:

|  |
| --- |
| d = (a = b + c) + r; // AVOID! |
| should be written as |
| a = b + c; |
| d = a + r; |

## Parentheses

* + - Use parentheses in expressions involving mixed operators to avoid operator precedence problems
    - Even if the operator precedence seems clear to you, it might not be to others-you shouldn't assume that other programmers know precedence as well as you do

|  |
| --- |
| if (a == b && c == d) // AVOID! |
| if ((a == b) && (c == d)) // RIGHT |

## Returning Values

* + - Try to make the structure of your program match the intent
    - Example:

|  |
| --- |
| if (booleanExpression) { |
| return true; |
| } else { |
| return false; |
| } |
| should instead be written as |
| return booleanExpression; |
| Similarly, |
| if (condition) { |
| return x; |
| } |
| return y; |
| should be written as |
| return (condition ? x : y); |

## Expressions before `?' in the Conditional Operator

* + - If an expression containing a binary operator appears before the? in the ternary ?: operator, it should be parenthesized. Example:
    - (x >= 0)? x : -x;

## Coding Example

|  |
| --- |
| /\* |
| \* Copyright (c) 2003-2004 Clover. |
| \* |
| \* All rights reserved. |
| \* |
| \* This software is the confidential and proprietary information of Clover |
| \* You shall not |
| \* disclose such Confidential Information and shall use it only in |
| \* accordance with the terms of the license agreement you entered into |
| \* with Clover |
| \*/ |
| package java.blah; |
|  |
| import java.blah.blahdy.BlahBlah; |
|  |
| /\*\* |
| \* Class description goes here. |
| \* |
| \* @version 1.82 02-Feb-2004 |
| \* @author Firstname Lastname |
| \*/ |
| public class Blah extends SomeClass { |
| /\* A class implementation comment can go here. \*/ |
|  |
| /\*\* classVar1 documentation comment \*/ |
| public static int classVar1; |
|  |
| /\*\* |
| \* classVar2 documentation comment that happens to be |
| \* more than one line long |
| \*/ |
| private static Object classVar2; |
|  |
| /\*\* instanceVar1 documentation comment \*/ |
| public Object instanceVar1; |
|  |
| /\*\* instanceVar2 documentation comment \*/ |
| protected int instanceVar2; |
|  |
| /\*\* instanceVar3 documentation comment \*/ |
| private Object[] instanceVar3; |
|  |
| /\*\* |
| \* ...constructor Blah documentation comment... |
| \*/ |
| public Blah() { |
| // ...implementation goes here... |
| } |
|  |
| /\*\* |
| \* ...method doSomething documentation comment... |
| \*/ |
| public void doSomething() { |
| // ...implementation goes here... |
| } |
|  |
| /\*\* |
| \* ...method doSomethingElse documentation comment... |
| \* @param someParam description |
| \*/ |
| public void doSomethingElse(Object someParam) { |
| // ...implementation goes here... |
| } |
| } |