
Supplementary of Sun Java Coding Conventions

Revision: 0.1

Author: Syd Chapman

Open Middleware Infrastructure Institute
Room 6005, Building 21 (Faraday)
Highfield Campus
University of Southampton
Hampshire, SO17 1BJ
Tel: +44 (0)2380 58787

Revision No.	Author	Changes Made	Date Revised
0.1	Syd Chapman	Added reference to Sun Java Coding Conventions	2708-04

Table of Contents

1 Introduction _____ 3

2 General checklist _____ 3

3 Additional Coding consideration _____ 4

1 Introduction

This is a supplementary document for Sun Java coding conventions. It is for all the developers who submit their codes to OMII.

2 General Checklist

The following checklist is intended to develop the quality of codes. Some items may not be relevant to all development. Any additional items identified in the course of reviews should be added.

Table 2.1 Product level

Guidelines
Pre-requisite and co-requisite for products and product levels
Application programming interfaces
Completion codes (abend codes)
Commands
Messages
Sample code
Operational Procedures

Table 2.2 General aspects above actual coding

Guidelines
Does the code completely and correctly implement the design?
Does code compile successfully?
Does the code build correctly?
Does compiled code run and get correct result?
Does code terminate once the correct result has obtained?
Have installation issues been considered?
Are all functions documented?
Is the code sufficiently traced?

Table 2.3 Coding Conventions

Guidelines
Are the Sun Java coding conventions followed? See OMII version derived from http://java.sun.com/docs/codeconv/ . Stored in BSCW
Is the copyright statement correct and present?
Does the code use any hard coded values (ie.port numbers, install/classpath, server names etc)?
The header(s) for the source code files should contain details of any included third party code along with dates of any changes to that included code and the author. This information is necessary for the OMII to keep track of licensing issues with Open Source
If the code is for general public consumption, are the Java Doc comments present, useful, up to date and relevant? Is the Java Doc available to review?
Do they follow the recommended guidelines? http://java.sun.com/j2se/javadoc/writingdoccomments/
Do variable and method names make the code self-documenting and promote understandability?
Do variable, class or method names exceed the compiler name length limits?
Are variables recycled and reused?

Do code comments explain why something is done or do they state the obvious?
Do the comments lie? Are they out of date?
Are change flags properly used?
Is there dead code or old commented out code littered about? Should it be deleted, or remain commented out?

3 Additional coding consideration for Sun Java coding standards

Apart from guidelines discussed in Section Two, there are more guidelines for Sun Java coding standards. Though Table 2.1 to Table 2.3 are also parts of Java coding standards, there is an additional coding consideration while developing Java-based software programs. Therefore, OMII have adopted Sun Java coding conventions. Our repository (URL:<http://www.omii.ac.uk>) has the related documents.

Java coding standards are intended to help all the developers developing and maintaining high quality codes. In return, this will enhance the overall code management process for all of us. Table 3.1 and Table 3.2 below can further supplement our Sun Java coding standards, and also supply further conventions for installing scripts.

If you find things you disagree with (they're wrong, unnecessary, or unclear), please talk to the author.

Table 3.1 Coding levels

Guidelines
Is the code clean and finished? Ensure there's no uncalled, unneeded, redundant code, or leftover stubs or test routines. Remove any unused variables that are redundant.
Does the source file contain a copyright statement?
Does the source file start with appropriate descriptive text?
Are all parameters clearly identified as such?
Are complex algorithms and code optimizations adequately commented? Complex areas, algorithms, and code optimizations should be sufficiently commented, so other developers can understand the code and walk through it.
Are all comments consistent with the code? Do the comments actually describe the code? Make sure the two match!
Is allocated memory freed? All allocated memory needs to be freed when no longer needed, Make sure memory is released in all code paths, especially in error code paths. Ensure memory is not freed more than once.
Are all objects (Database connections, sockets, file handles etc) freed when no longer required. Check especially that error code paths are freeing up objects that are no longer required. Ensure these objects are not freed more than once.
Are all variable properly defined with meaningful, consistent and clear names?

Table 3.2 **Java specific**

Guidelines
Does each class have appropriate constructors and destructors?
Do any subclasses have common members that should be in the super-class?
Can the class inheritance hierarchy be simplified?
Are the right abstractions used?
Are encapsulation principles violated?
Do exceptions include enough information to communicate what failed and why?
Are error-conditions propagated, logged or ignored?
Is a meaningful message emitted? Messages should include what went wrong, with relevant clues why it went wrong, and a possible user action, or a useful exception. Where exceptions are used by logging, sufficient information is required to locate and understand the situation.
Do entry logs include input parameters?
Do exit logs include return values? Are entry and exit logs properly paired?
Do major branches or decision points in the code include trace messages to indicate flow of control?
Do exceptions include enough information to communicate what failed and why?
Do empty catch blocks have at least a comment to say why it's OK to ignore them.
Is there any information loss about failure reason when exceptions are mapped and re-thrown?
Are exceptions handled in the right part of the code?