**Java Coding Conventions and Development Standards**

Version 1.1

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Revision Notes** |
| 1.0 | Dec 2015 | Atul Shelke | Initial Version. |
| 1.1 | 5 Jan 2018 | Rabi Sahu  Sitaram Kamble | Updated & formatted |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

1. Introduction 4

1.1 Scope 4

2. Java Coding Standards and Best Practices Links 4

3. Review Severity 4

4. Code Review Guidelines 5

4.1 Code Formatting and Documentation 5

4.2 Exception Handling and Logging 5

4.3 Test Cases and Code Coverage 5

4.4 Naming conventions and Coding Styles follow Standard Practices 6

4.5 Reusability and Redundant Code 6

4.6 Security 6

4.7 Performance and Scalability 6

4.8 Adherence to the Design 7

# Introduction

It is always our effort to provide clean and consistent codebase, and to be productive as a development team. This document provides general coding standards to be followed in Development using Java based technologies (e.g. Spring).

It is required to share and to be in agreement with client before implementing these coding standards, and if there are any client provided guidelines, they should override the guidelines in this document.

## Scope

This document provides various references to industry standard Java Coding conventions.

# Java Coding Standards and Best Practices Links

The original standards document from Oracle on coding standards is at below link. We recommend following original guidelines, unless if any are obsolete in the new version

<http://www.oracle.com/technetwork/java/codeconvtoc-136057.html>

<http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

The updated guidelines can be found at

<https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java>

<https://docs.oracle.com/cd/A97688_16/generic.903/bp/java.htm>

<https://docs.oracle.com/cd/A97688_16/generic.903/bp/j2ee.htm>

<https://docs.oracle.com/cd/A97688_16/generic.903/bp/framework.htm>

<http://www.javacodegeeks.com/2015/06/java-programming-tips-best-practices-beginners.html>

<http://www.javapractices.com/home/HomeAction.do>

<http://archive.oreilly.com/topics/java/Java_Design>

Also there are some other source of links can be found here:

<https://google.github.io/styleguide/javaguide.html>

<http://geosoft.no/development/javastyle.html>

<https://wiki.sei.cmu.edu/confluence/display/java/Java+Coding+Guidelines>

# Review Severity

The table belows provides severity for general code review areas that we focus on while doing java code review.

|  |  |  |
| --- | --- | --- |
| Sr.No. | Areas | Severity |
| 1 | Naming Conventions and Coding style | Medium |
| 2 | Control Structures and Logical issues | Medium/High |
| 3 | Redundant Code | Medium |
| 4 | Performance Issues | High |
| 5 | Security Issues | High |
| 6 | Scalability Issues | High |
| 7 | Functional Issues | High |
| 8 | Error and Exception Handling | High |
| 9 | Logging | High |
| 10 | Reusability | Medium |
| 11 | Unit Test and Code Coverage | High |
| 12 | Documentation | Medium |
| 13 | Code duplicacy | Medium |

# Code Review Guidelines

The focus should be to use automated static code review and design review tool as much as possible over manual code review.

The following check list is being used during code review

<https://dzone.com/articles/java-code-review-checklist>

Below are some additional guidelines for better code quality

## Code Formatting and Documentation

* Verify class and method level comments added are comprehensible and add value to the maintainability of the code. Mainly the comments should provide what is purpose of the code, than what it is doing.
* Verify Comments are neither too numerous nor too verbose.
* Inline comments and descriptive blocks are added wherever necessary. Write inline comments mainly for complex logic implementation.
* Verify if copyright notices / disclaimers if applicable are added in the source files.
* Approved code formatter for the project is configured into IDE and followed for code formatting.
* Verify if the use and function of third-party libraries is documented.

## Exception Handling and Logging

* Verify there are no classes that throw exceptions and exceptions are dealt with in the class itself.
* Catch exception only when right treatment can be given to the exception, otherwise do not catch. Do not just catch and log the exceptions.
* Appropriate logging framework is used and logging is added at pre-defined places in the required format. Logging and exception handling are in line with the logging and exception handling framework standardized at the project level.
* Business level user defined exceptions are only thrown at Module levels.

## Test Cases and Code Coverage

* Unit Test cases for every class including every (public) method are created.
* Code coverage reports are generated and meet the required KPI criteria.
* Does unit tests actually test that the code is performing the intended functionality?
* Cobertura code coverage reports are generated for functional testing coverage.
* Spring test and Mockito like tools are used during unit test cases. Singletons are tested thoroughly.
* Validate both Line and Branch level Code coverage reports generated.

## Naming conventions and Coding Styles follow Standard Practices

* Verify that java naming standards are used correctly.
* No unwanted / unjustifiable static methods OR blocks used.
* Code Compiles to the standards and no dependencies are local. Verify that the code is built with appropriate integration and build tools pointing to the correct repositories.
* Appropriate versions are used for all dependencies to build the code.
* Types have been Generalized wherever possible and parameterized types have been used appropriately
* Business logic is confined at the service layer and not included in the UX/Presentation tier. Interface is defined for each service implementation.
* PMD, Checkstyles and Sonar violations are turned ON and the code does not show any violations.
* Verify Formatter, Checkstyles, PMD, Sonar rule files are latest checked out from the project repositories and no local copies are used.

## Reusability and Redundant Code

* Verify that any business logic code is not repeated and should be packaged to appropriate parent classes. Or even separate package/project.
* Repetitive code has been factored out.

## Security

* Are all data inputs checked (for the correct type, length, format, and range) and encoded?
* Where third-party utilities are used, are returning errors being caught?
* No sensitive data is logged or thrown as part of exception.
* Are output values checked and encoded?
* Are invalid parameter values handled?
* Any NFRS defined for security are handled and security concerns have been addressed?

## Performance and Scalability

* Potential threading issues have been eliminated where possible.
* Caching of Master Data is done.
* Batch commit is used to improve database operation performance.
* Choice of using Stored Procedures wherever required and SQLs is appropriately made.
* No calls within a loop that will cause memory issues.
* Chunking is used when loading, reading data on screen, from database.

## Adherence to the Design

* Controller patterns for microservices are strictly followed and appropriate java singleton patterns for Controllers is followed.
* Appropriate demarcation in the code in the controller and the delegators classes. Verify that there is no continuation of any logic across layers.
* Utility classes are identified and packaged correctly. Utility jars are identified across the common service layer and UX/Presentation layer codes.
* Business code is strictly independent and is tested without having any dependency on the UI or Controller / delegator classes.