Code Review Checklist

Approach & Communications

- Do you ask questions before making statements?
- Do you avoid the “Why” questions and use something like “what was the reason behind that decision”?
- Do you ensure that discussion stays focused on the code, not the coder?
- Do you praise good code?
- Do you take into account that several solutions are usually possible?
- Do you enforce your own opinion?
- Do you explain your suggestions politely?
- Do you use words such as “in my opinion”, “I think that” instead of “this is wrong” and “we should do this”?
- Do you review less than 200-400 lines of code at once?
- Do you treat code review process as positive and effective way to minimize costs and increase quality?
- Do you use code review results as the only way to assess developers?
- Do you criticize publicly?
- Do you praise publicly?
- Do you practice pair review and discussions?

Common Questions

- Do you understand the code?
- Can the code be refactored to make it clearer?
- Are there test cases for code changes?
- Do all changes implement a required feature, or fix a bug that is listed in the change description?
- Does the code adhere to the Coding Standards?
    - Indentation
    - Naming
    - Bracket style
    - Etc.
- Do you understand the design?
- Does the implementation match the design?
- Does the code build without errors and warnings?
- Are there Sonar warnings for the new code?
- Is the new code “green” in Idea?
- Does the code fail fast in presence of unrecoverable errors?
- Is objects lifecycle well understood and managed?
- Could we have used existing libraries and tools?

Code Documentation

- Are the comments necessary?
- Are the comments accurate?
- Are variable names spelled correctly and consistently?
- Complex algorithms should be thoroughly commented.
- Code that has been optimized or modified to “work around” an issue should be thoroughly commented, so as to avoid confusion and re-introduction of bugs.
- Code that has been “commented out” should be explained or removed.
- Code that needs to be reworked should have a TODO comment and a clear explanation of what needs to be done (and why it was not done at this iteration).
- All public members should have Javadocs – refer to the code standard and contract documentation standard.
- Absence of automatically generated comments
- Motivation in design decisions should be explained.

Error Handling

- Are method arguments validated and rejected with an exception if they are invalid?
- Is the code fault-tolerant? Is it error-tolerant?
- Will the code handle abnormal conditions or malformed input?
- Does the code fail gracefully if it encounters an unexpected condition?
- Are assertions used to verify assumptions about the functioning of the code?
- Are exceptions used to indicate error rather than returning status or error codes?
- Are finally blocks used for code that must execute following a try?
- Does the code simply catch exceptions and log them?

- Does the code catch general exception (java.lang.Exception)?

Abstraction and design

- Is code written against an interface rather than an implementing class?
- Is "instanceof" used for possibly incorrect downcasts?
- Does the code uses static variables? Is their use justified?
- Is the code as generalized/abstracted as it could be? Is the code a candidate for reusability?
- Are there duplicates in the code?
- Are all methods in a class as short as they could be?
- Do methods and variables have descriptive names?
- Are methods doing only one thing (actually the thing that their name says it does)?
- Are variables/methods declared with the narrowest scope possible?
- Is aggregation preferred to inheritance?
- Is the Demeter's law violated (http://en.wikipedia.org/wiki/Law_of_Demeter)?
- Are side-effects localized?
- Are functions pure (if possible)?
- Is there any state that changes the way object behaves?
- Is there any non-trivial temporal cohesion (methods such as init()) ?
- Is order of operations significant?
- Could a well-known pattern be applied to the problem?
- Do we really need to apply this pattern here?
- Is LSP violated? Can the parent object be substituted for any child object without undesired effects?
- Is OCP violated (can we add behavior without the need to modify existing code)?
- Is dependency injection used?
- Is the data encapsulated? Is defensive copying used?
- Do we use a proper stereotype (such as "value object" or "repository")?
- Does the code enforce invariants using assertions?

Debugging and maintenance

- Do classes override toString?
- Are there necessary monitoring and JMX hooks?
- Is there enough logging?
- Is logging too verbose?
- Is logging level selected correctly?

General code quality

- If equals is overridden, is it done correctly?
- Does the code release resources?
- Does the code release resources more than once?

Security

- Is the code vulnerable to unauthorized access or malicious use or modification?
- Does the code accept user's input without validation or sanitization?
- Does the code make any business decisions based on user's input? Are these decisions safe?
- Does the code output sensitive information (such as passwords) in unencrypted way?
- Is the code vulnerable to attacks?

Thread safety

- Does the code have proper synchronization?
- Are all the paths in the code use the same synchronization?
- Is double-checked locking idiom employed correctly?
- Do we need synchronization here?
- Can we use volatiles or atomics?
- Is the synchronization state and mechanics available for client code?
- Do we correctly use non-thread-safe classes from JDK (NumberFormat etc.)?
- Could we have used tools from java.util.concurrent?

Performance

- Does the code make efficient use of memory, CPU cycles, bandwidth or other system resources?
- Does the code use the most efficient class when dealing with certain resources?
- How frequent will the code be called? Do we need to pay attention to garbage allocation?
- Is there enough performance metrics collected?
- Does the code make use of Profiler?
- Can a more effective algorithm be used?