Java Code Inspection Checklist

1. Variable and Constant Declaration Defects (VC)

1. Are descriptive variable and constant names used in accord with naming conventions?
2. Are there variables with confusingly similar names?
3. Is every variable properly initialized?
4. Could any non-local variables be made local?
5. Are there literal constants that should be named constants?
6. Are there macros that should be constants?
7. Are there variables that should be constants?

2. Function Definition Defects (FD)

8. Are descriptive function names used in accord with naming conventions?
9. Is every function parameter value checked before being used?
10. For every function: Does it return the correct value at every function return point?

3. Class Definition Defects (CD)

11. Does each class have an appropriate constructor and destructor?
12. For each member of every class: Could access to the member be further restricted?
13. Do any derived classes have common members that should be in the base class?
14. Can the class inheritance hierarchy be simplified?

4. Computation/Numeric Defects (CN)

15. Is overflow or underflow possible during a computation?
16. For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?
17. Are parentheses used to avoid ambiguity?

5. Comparison/Relational Defects (CR)

18. Are the comparison operators correct?
19. Is each boolean expression correct?
20. Are there improper and unnoticed side-effects of a comparison?

6. Control Flow Defects (CF)

21. For each loop: Is the best choice of looping constructs used?
22. Will all loops terminate?
23. When there are multiple exits from a loop, is each exit necessary and handled properly?

24. Does each switch statement have a default case?
25. Are missing switch case break statements correct and marked with a comment?
26. Is the nesting of loops and branches too deep, and is it correct?
27. Can any nested if statements be converted into a switch statement?
28. Are null bodied control structures correct and marked with braces or comments?
29. Does every function terminate?
30. Are goto statements avoided?

7. Input-Output Defects (IO)

31. Have all files been opened before use?
32. Are the attributes of the open statement consistent with the use of the file?
33. Have all files been closed after use?
34. Is buffered data flushed?
35. Are there spelling or grammatical errors in any text printed or displayed?
36. Are error conditions checked?

8. Module Interface Defects (MI)

37. Are the number, order, types, and values of parameters in every function call in agreement with the called function’s declaration?
38. Do the values in units agree (e.g., inches versus yards)?

9. Comment Defects (CM)

39. Does every function, class, and file have an appropriate header comment?
40. Does every variable or constant declaration have a comment?
41. Is the underlying behavior of each function and class expressed in plain language?
42. Is the header comment for each function and class consistent with the behavior of the function or class?
43. Do the comments and code agree?
44. Do the comments help in understanding the code?
45. Are there enough comments in the code
46. Are there too many comments in the code?

10. Packaging Defects (LP)

47. For each file: Does it contain only one class?
48. For each function: Is it no more than about 60 lines long?
49. For each class: Is no more than 2000 lines long (Sun Coding Standard) ?

11. Modularity Defects (MO)

50. Is there a low level of coupling between packages (classes)?
51. Is there a high level of cohesion within each package?
52. Is there duplicate code that could be replaced by a call to a function that provides the behavior of the duplicate code?
53. Are framework classes used where and when appropriate?

12. Performance Defects (PE) [Optional]

54. Can better data structures or more efficient algorithms be used?
55. Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?
56. Can the cost of recomputing a value be reduced by computing it once and storing the results?
57. Is every result that is computed and stored actually used?
58. Can a computation be moved outside a loop?
59. Are there tests within a loop that do not need to be done?
60. Can a short loop be unrolled?
61. Are there two loops operating on the same data that can be combined into one?

slightly adapted from the C++ Inpsection Checklist of Christopher Fox,
http://www.cs.jmu.edu/users/foxcj/cs555/StdDoc/CppChk.htm.