markConstraint: respected?

merge: respected?

checkConstraint: respected?

**Naming Conventions**

1. All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.

markConstraint: respected?

Line 197: name can be named as variableName, will be more meaningful

Line 204, variable old can be named as oldConstraintText, will be more meaningful

Line 198, for consistency can be named “info” (the result of “varInfos.get()” is stored in a variable called“info” (or containing this noun) in all other methods) [To check]

merge: respected?

Line 221: name can be named as variableName, will be more meaningful

checkConstraint: respected?

yes

2. If one-character variables are used, they are used only for temporary “throwaway” variables, such as those used in for loops.

markConstraint: respected

yes

merge: respected?

yes

checkConstraint: respected?

yes

3. Class names are nouns, in mixed case, with the first letter of each word in capitalized. Examples: class Raster; class ImageSprite;

Yes in all the file VariableTable.java

4. Interface names should be capitalized like classes.

No interfaces found in the source file

5. Method names should be verbs, with the first letter of each addition word capitalized. Examples: getBackground(); computeTemperature().

markConstraint: respected

yes

merge: respected?

Yes

checkConstraint: respected?

yes

6. Class variables, also called attributes, are mixed case, but might begin with an underscore (‘\_’) followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized. Examples: \_windowHeight, timeSeriesData.

Yes in all file

7. Constants are declared using all uppercase with words separated by an underscore. Examples: MIN\_WIDTH; MAX\_HEIGHT;

Yes respected in all file (Verify line 99-100-101)

***Indention***

<http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-136091.html>

8. Three or four spaces are used for indentation and done so consistently

markConstraint: respected?

Line 209 not respected (uses 5x 4spaces and 2 spaces)

If using a “indent 8 char in expressions”, is more than that.

Is a conventional intendation (aligned with the first argument)

Since is a method call, can be a multiple of 8 char, but on line 208 I’m on the right margin, so I have to indent 8 char accordingly to oracle link

merge: respected?

Yes

checkConstraint: respected?

Line 291 and 307 not respected (uses 5x 4spaces and 2 spaces)

Same for markContraint

9. No tabs are used to indent

Yes respected in all the file

***Braces***

10. Consistent bracing style is used, either the preferred “Allman” style (first brace goes underneath the opening block) or the “Kernighan and Ritchie” style (first brace is on the same line of the instruction that opens the new block).

Yes respected in all the file

11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces. Example:

Avoid this:

if ( condition )

doThis();

Instead do this:

if ( condition )

{

doThis();

}

markConstraint: respected?

No, line 199 is not respecting this

merge: respected?

Yes respected

checkConstraint: respected?

Yes respected

***File Organization***

12. Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).

Line 50 and 52 space not needed? Check

Line 99-101 space needed? check

yes

13. Where practical, line length does not exceed 80 characters.

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

Yes

14. When line length must exceed 80 characters, it does NOT exceed 120 characters.

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

Yes

**Wrapping Lines**

15. Line break occurs after a comma or an operator.

markConstraint: respected?

Line 200, ( is not used as an operator (is a part of the function)., there is a line break

Is preferred to be:

throw new JDOFatalInternalException(I18NHelper.getMessage(messages,

"jqlc.variabletable.markconstraint.varnotfound", //NOI18N

name));

or

throw new JDOFatalInternalException(

I18NHelper.getMessage(messages,

"jqlc.variabletable.markconstraint.varnotfound", //NOI18N

name));

Accordingly to oracle suggestions

merge: respected?

Line 247 ( is not used as an operator (is a part of the function).

Must indent 8 chars not 4.

checkConstraint: respected?

Line 289 (is not an operator (is a part of the method)..

Not a method declaration, the indentation must be of 8 chars, not aligned to “messages”

Line 305 ( is not an operator.

The same of line 289

16. Higher-level breaks are used.

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

Yes

17. A new statement is aligned with the beginning of the expression at the same level as the previous line.

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

Yes

**Comments**

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

VariableTable class is not sufficiently commented, there is the license but no explanation of what the hell is doing this class.

VarInfo is commented

markConstraint: respected?

“The method sets the constraint field of the VarInfo object to true.”

Actually, set the constraint field equal to the expr given.

Blocks of code can be commented more adequately (explanation of the old variable?)

merge: respected?

Method yes

Blocks yes

checkConstraint: respected?

Method not commented

Line of code not too much:

The blocks inside the switch are commented but no other comments are present.

19. Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.

markConstraint: respected?

merge: respected?

checkConstraint: respected?

YES in all, no commented lines of code

**Java Source Files**

20. Each Java source file contains a single public class or interface.

Yes

21. The public class is the first class or interface in the file.

Yes

22. Check that the external program interfaces are implemented consistently with what is described in the javadoc.

markConstraint: respected?

merge: respected?

checkConstraint: respected?

No implementation of interface

23. Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

No documentation specified for this method

Javadoc is incomplete!

**Package and Import Statements**

24. If any package statements are needed, they should be the first non-comment statements. Import statements follow.

Yes is the first not commented line (there is a space for separating the comments and the statement.

All the import statement follow the package statement

**Class and Interface Declarations**

25. The class or interface declarations shall be in the following order:

A. class/interface documentation comment

Yes (but no documentation comment, only author and version)

B. class or interface statement

yes

C. class/interface implementation comment, if necessary

Not present

D. class (static) variables

a. first public class variables

b. next protected class variables

c. next package level (no access modifier)

d. last private class variables

E. instance variables

a. first public instance variables

e. next protected instance variables

f. next package level (no access modifier)

g. last private instance variables

yes, there is also a class- in-class

F. constructors

yes

G. methods

Yes

Document structure is respected.

26. Methods are grouped by functionality rather than by scope or accessibility.

Yes respected

27. Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

markConstraint: respected?

Yes, but it directly modify the entry.constraint value.

Is not a problem since the fact the class has only a constructor method, but a getter and setter can be implemented for better coupling, but for now is adequate.

merge: respected?

Yes, but it directly modify the entry.constraint value. (copupling is ok)

checkConstraint: respected?

Direct access to VarInfo class variables, but ok.

Due the fact there are direct access to non-constant fields, is better to implement getters and setters. (is a tight coupling)

**Initialization and Declarations**

28. Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected)

Yes in all

29. Check that variables are declared in the proper scope

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

Yes

30. Check that constructors are called when a new object is desired

markConstraint: respected?

merge: respected?

checkConstraint: respected?

No new objects desired.

31. Check that all object references are initialized before use

markConstraint: respected?

merge: respected?

checkConstraint: respected?

Yes in all, all the references are initialized in the constructor (all overrides )

32. Variables are initialized where they are declared, unless dependent upon a computation

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

Yes

33. Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces “{“ and “}” ). The exception is a variable can be declared in a ‘for’ loop.

markConstraint: respected?

Line 203: String old is not at the beginning of a block

merge: respected?

Yes

checkConstraint: respected?

Yes

**Method Calls**

34. Check that parameters are presented in the correct order

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

Yes

35. Check that the correct method is being called, or should it be a different method with a similar name

markConstraint: respected?

merge: respected?

checkConstraint: respected?

Yes all

36. Check that method returned values are used properly

markConstraint: respected?

merge: respected?

checkConstraint: respected?

Yes all

**Arrays**

37. Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index)

markConstraint: respected?

merge: respected?

checkConstraint: respected?

Yes, no off-by-one errors; merge uses iterator in a correct way

38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds

markConstraint: respected?

merge: respected?

checkConstraint: respected?

Yes, merge uses iterator in a correct way (tehe same of 37)

39. Check that constructors are called when a new array item is desired

markConstraint: respected?

merge: respected?

checkConstraint: respected?

No methods need new array item and the constructor of each array is called in each object constructor.

**Object Comparison**

40. Check that all objects (including Strings) are compared with "equals" and not with "=="

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

Yes

The only non “equals” comparison is made if the object is null (but obviously is the result of the verInfos.get() call that is null if no object found)

**Output Format**

41. Check that displayed output is free of spelling and grammatical errors

markConstraint: respected?

merge: respected?

checkConstraint: respected?

No output displayed

42. Check that error messages are comprehensive and provide guidance as to how to correct the problem

markConstraint: respected?

merge: respected?

checkConstraint: respected?

There are exceptions with message and various infos

43. Check that the output is formatted correctly in terms of line stepping and spacing

markConstraint: respected?

merge: respected?

checkConstraint: respected?

No output displayed

**Computation, Comparisons and Assignments**

44. Check that the implementation avoids “brutish programming: (see http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html)

markConstraint: respected?

merge: respected?

checkConstraint: respected?

merge and checkConstraint both refers to a variable define in the “VarInfo” class that defines a finite set of named constants.

Those constant (line 99-101) can be set in a Enum.

45. Check order of computation/evaluation, operator precedence and parenthesizing

markConstraint: respected?

merge: respected?

checkConstraint: respected?

All alright

46. Check the liberal use of parenthesis is used to avoid operator precedence problems.

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

Yes

47. Check that all denominators of a division are prevented from being zero

markConstraint: respected?

merge: respected?

checkConstraint: respected?

No divisions, so YES for all

48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding

markConstraint: respected?

merge: respected?

checkConstraint: respected?

No arithmetic operations, so Yes respected for all

49. Check that the comparison and Boolean operators are correct

markConstraint: respected?

merge: respected?

checkConstraint: respected?

Yes in all

50. Check throw-catch expressions, and check that the error condition is actually legitimate

markConstraint: respected?

merge: respected?

checkConstraint: respected?

Yes for all. -> attach the explanation that is written in exceptions

51. Check that the code is free of any implicit type conversions

markConstraint: respected?

Yes

merge: respected?

Yes

checkConstraint: respected?

Yes

**Exceptions**

52. Check that the relevant exceptions are caught

markConstraint: respected?

JDOFatalInternalException is thrown. In some classes is declared to be thrown (like ErrorMsg), but here not. It’s not a checked exception (extends RuntimeException) so it’s not mandatory, but is also not specified in the javadoc

[JDOUnsupportedOptionException](http://glassfish.pompel.me/com/sun/jdo/api/persistence/support/JDOUnsupportedOptionException.html) also is thrown by the line 207, but its nos specified anywhere

Obviously these exception are not thrown if the variable exists and if the variable exists only once, so it’s normal that are unchecked exception

yes

merge: respected?

yes

checkConstraint: respected?

Yes

The above analysis is for all the exception thrown in the analyzed methods

53. Check that the appropriate action are taken for each catch block

No catch block, so yes

**Flow of Control**

54. In a switch statement, check that all cases are addressed by break or return

markConstraint: respected?

No switch, yes

merge: respected?

On line 289 there is no “return” or “break”, but is an exception so the flow of the block is blocked

checkConstraint: respected?

No switch, yes

55. Check that all switch statements have a default branch

markConstraint: respected?

No switch, yes

merge: respected?

Switch on line 281, no default branch

checkConstraint: respected?

No switch, yes

56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions

markConstraint: respected?

No loop, yes

merge: respected?

The loop of line 219 is correctly initialized and has the therminal exception. The incrementation is done inside the block of the loop and not in the declaration.

checkConstraint: respected?

No loop, yes

**Files**

57. Check that all files are properly declared and opened

No file

58. Check that all files are closed properly, even in the case of an error

No file

59. Check that EOF conditions are detected and handled correctly

No file

60. Check that all file exceptions are caught and dealt with accordingly

No file

OTHER PROBLEMS:

Line 204, entry.constraint==null can be set as entry.constraint == null, to be “uniforme allo stile usato in ogni comparazione”