Politecnico di Milano
School of Industrial and Information Engineering
Master Course in Computer Science and Engineering
DEIB Department

Code Inspection

4th January 2016

Moreno SARDELLA - 859239

Academic Year 2015–2016

Contents

Introduction

0.1 Assigned Classes

Project properties:
Group Id: org.glassfish.main.persistence.cmp
Artifact Id: cmp-support-sqlstore
Version: 4.1.1
Packaging: glassfish-jar
Name: support-sqlstore module for cmp

Assigned class:
com.sun.jdo.spi.persistence.support.sqlstore.sql.generator.SelectQueryPlan.java

Assigned method:
processOrderConstraints( )

0.2 Functional role of assigned set of classes

«This class prepares the generation of select statements, by joining the constraint stacks of all retrieve descriptors into one stack.» (Javadoc)

Chapter 1

Issues

1.1 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.***

```
1458: if ((status & ST_BUILT) > 0 || (status &
   ST_OC_BUILT) > 0) {
```

ST_ BUILT and *ST_ OC_ BUILT* constants have partial meaningful names. Probably *OC* stand for «operation constraint»..

```
1473: ConstraintNode opNode = (ConstraintNode)
   constraint.stack.get(i);
```

opNode could be refactored as *operationNode*.

```
1498: ConstraintFieldDesc consFieldDesc = null;
```

consFieldDesc could be refactored as *constraintFieldDesc*.

```
1545: ArrayList oa = (ArrayList) orderByArray.get(j);
```

oa has meaningless name.

```
1550: ConstraintFieldDesc ob = (ConstraintFieldDesc) oa
   .get(k);
```

ob has meaningless name.

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

   Fullfilled point.

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

```
77: public class SelectQueryPlan extends QueryPlan {
```

   SelectQueryPlan could be refatored as *QueryPlanSelector*.

4. ***Interface names should be capitalized like classes.***

   There are no intefaces.

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

   Fullfilled point.

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.***

```
93: protected Constraint constraint;
[...]
96: public int options;
[...]
99: private Iterator fieldIterator;
[...]
105: private int aggregateResultType;
[...]
111: private ArrayList foreignPlans;
[...]
117: protected ForeignFieldDesc parentField;
[...]
123: private boolean prefetched;
[...]
125: private Concurrency concurrency;
[...]
```

```
128: private BitSet hierarchicalGroupMask;
[...]
131: private BitSet independentGroupMask;
[...]
134: private BitSet fieldMask;
[...]
136: private Map foreignConstraintPlans;
[...]
138: private ResultDesc resultDesc;
[...]
146: private boolean appendAndOp;
```

All class variables do not begin with an underscore.

7. ***Constants are declared using all uppercase with words separated by an underscore. Examples: MIN_WIDTH; MAX_HEIGHT;***

```
149: private final static Logger logger =
   LogHelperSQLStore.getLogger();
```

logger could be refatored as LOGGER.

1.2 Indention

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

9. ***No tabs are used to indent***

1.3 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).***

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(); }***

1.4 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).

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

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

1.5 Wrapping Lines

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

16. Higher-level breaks are used.

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

1.6 Comments

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

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.

1.7 Java Source Files

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

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

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

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

1.8 Package and Import Statements

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

1.9 Class and Interface Declarations

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

- *A. class/interface documentation comment*
- *B. class or interface statement*
- *C. class/interface implementation comment, if necessary*
- *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*
- *F. constructors*
- *G. methods*

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

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

1.10 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)

29. Check that variables are declared in the proper scope

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

31. Check that all object references are initialized before use

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

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.

1.11 Method Calls

34. Check that parameters are presented in the correct order

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

36. Check that method returned values are used properly

1.12 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)

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

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

1.13 Object Comparison

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

1.14 Output Format

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

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

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

1.15 Computation, Comparisons and Assignments

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

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

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

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

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

49. Check that the comparison and Boolean operators are correct

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

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

1.16 Exceptions

52. Check that the relevant exceptions are caught

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

1.17 Flow of Control

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

55. Check that all switch statements have a default branch

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

1.18 Files

57. Check that all files are properly declared and opened

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

59. Check that EOF conditions are detected and handled correctly

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

Chapter 2

Other Problems

Appendix A

Document Information

A.1 Effort

Approximately **10 hours** have been spent making this document.

A.2 Tool Used

- **LyX**: www.lyx.org
- **NetBeans 8.1 - Findbugs plugin**: https://netbeans.org/kb/docs/java/code-inspect.html
- **NetBeans 8.1 - Checkstyle plugin**: http://plugins.netbeans.org/plugin/3413/checkstyle-beans
- **SonarQube**: http://docs.sonarqube.org/

Appendix B

References

- Code Inspection Checklist:
  https://dl.dropboxusercontent.com/u/79082424/Assignment%203.pdf