Politecnico di Milano

School of Industrial and Information Engineering

Master Course in Computer Science and Engineering

DEIB Department

Code Inspection

10th January 2016

Moreno SARDELLA - 859239

Academic Year 2015–2016

Contents

Introduction

0.1 Assigned Code

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;***

   ```
   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.***

   ```
   protected Constraint constraint;
   [...]
   public int options;
   [...]
   private Iterator fieldIterator;
   [...]
   private int aggregateResultType;
   [...]
   private ArrayList foreignPlans;
   [...]
   protected ForeignFieldDesc parentField;
   [...]
   private boolean prefetched;
   [...]
   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***

   Fullfilled point.

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

   Fullfilled point.

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).***

    Fullfilled point.

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(); }

Fullfilled point.

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).***

    Fullfilled point.

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

```
1475: if ((opNode instanceof ConstraintOperation)
1476:       && ((((ConstraintOperation) opNode).
   operation == ActionDesc.OP_ORDERBY) ||
1477:       (((ConstraintOperation) opNode).operation
   == ActionDesc.OP_ORDERBY_DESC))) {
1478:   pos = -1;
1479:   if ((i > 1) && (constraint.stack.get(i - 2)
   instanceof ConstraintValue)) {
1480:       pos = ((Integer) ((ConstraintValue)
   constraint.stack.get(i - 2)).getValue() ).intValue()
   ;
[...]
1528: if (((ConstraintOperation) opNode).operation ==
   ActionDesc.OP_ORDERBY_DESC) {
```

Lines 1476, 1477, 1479, 1480 and 1528 could be refactored using new boolean variables.

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

    Fullfilled point.

1.5 Wrapping Lines

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

```
1476: && ((((ConstraintOperation) opNode).operation ==
    ActionDesc.OP_ORDERBY) ||
1477: (((ConstraintOperation) opNode).operation ==
    ActionDesc.OP_ORDERBY_DESC))) {
```

Line 1476 is broken after an OR..

16. Higher-level breaks are used.

```
1475: if ((opNode instanceof ConstraintOperation)
1476:       && ((((ConstraintOperation) opNode).
    operation == ActionDesc.OP_ORDERBY) ||
1477:       (((ConstraintOperation) opNode).operation
    == ActionDesc.OP_ORDERBY_DESC))) {
[...]
1511:       throw new JDOUserException(I18NHelper.
    getMessage(messages,
1512:                "core.generic.notinstanceof", //
    NOI18N
1513:                fieldDesc.getClass().getName(),
1514:               "LocalFieldDesc")); // NOI18N
[...]
1522:        throw new JDOUserException(I18NHelper.
    getMessage(messages,
1523:               "core.generic.notinstanceof", //
    NOI18N
1524:                fieldNode.getClass().getName(),
1525:                "ConstraintFieldName/
    ConstraintFieldDesc")); // NOI18N
```

Lower-level breaks are used in lines between 1475 and 1477, 1511 and 1514, and also between 1522 and 1525.

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

Fullfilled point.

1.6 Comments

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

    Fullfilled point.

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.***

    There is no commented out code.

1.7 Java Source Files

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

    Fullfilled point.

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

    Fullfilled point.

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

    Fullfilled point.

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

1.8 Package and Import Statements

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

    Fullfilled point.

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

```
private static final int ST_JOINED = 0x2;
[...]
public static final int ST_C_BUILT = 0x4;
[...]
public static final int ST_OC_BUILT = 0x10;
[...]
protected Constraint constraint;
[...]
public int options;
[...]
private Iterator fieldIterator;
[...]
    private int aggregateResultType;
[...]
    private ArrayList foreignPlans;
[...]
    protected ForeignFieldDesc parentField;
[...]
    private boolean prefetched;
[...]
    private Concurrency concurrency;
[...]
    private BitSet hierarchicalGroupMask;
```

```
[...]
    private BitSet independentGroupMask;
[...]
    private BitSet fieldMask;
[...]
    private Map foreignConstraintPlans;
[...]
    private ResultDesc resultDesc;
[...]
    private boolean appendAndOp;
[...]
    private final static Logger logger =
  LogHelperSQLStore.getLogger();
[...]
    public static final String
  MULTILEVEL_PREFETCH_PROPERTY =
        "com.sun.jdo.spi.persistence.support.
  sqlstore.MULTILEVEL_PREFETCH"; // NOI18N
[...]
    private static final boolean
  MULTILEVEL_PREFETCH = Boolean.valueOf(
        System.getProperty(
  MULTILEVEL_PREFETCH_PROPERTY, "false")).
  booleanValue(); // NOI18N
```

Class variables declaration do not fullfill the point.

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

    Fullfilled point.

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

    The assigned method *processOrderConstraints* is too long (110 lines) . It could be refactorized.

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)***

Fullfilled point.

29. ***Check that variables are declared in the proper scope***

Fullfilled point.

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

31. ***Check that all object references are initialized before use***

Fullfilled point.

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

Fullfilled point.

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.***

```
if (orderByArray.get(insertAt) == null) {
    orderByArray.set(insertAt , new ArrayList());
}

ConstraintNode fieldNode = (ConstraintNode) constraint.stack.get(i - 1);
ConstraintFieldDesc consFieldDesc = null
[...]
if (((ConstraintField) fieldNode).originalPlan != null) {
    originalPlan = ((ConstraintField) fieldNode).originalPlan;
}

FieldDesc fieldDesc = originalPlan.config.
   getField(((ConstraintFieldName) fieldNode).name);
[...]
if (((ConstraintOperation) opNode).operation == ActionDesc.OP_ORDERBY_DESC) {
    consFieldDesc.ordering = -1;
}
```

```
1531:
1532: // Remember constraint in orderByArray.
1533: ArrayList temp = (ArrayList) (orderByArray.get(
    insertAt));
```

Lines 1497,1498, 1504, 1505 and 1533 do not fullfill the point.

1.11 Method Calls

34. Check that parameters are presented in the correct order

The assigned method does not have parameters.

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

Fullfilled point.

36. Check that method returned values are used properly

The assigned method does not return any value.

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)

Fullfilled point.

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

Fullfilled point.

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

Fullfilled point.

1.13 Object Comparison

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

```
1493: if (orderByArray.get(insertAt) == null) {
[...]
1501: if (((ConstraintField) fieldNode).originalPlan !=
    null) {
[...]
1547: if (constraint == null) {
```

Lines 1493, 1501 and 1547 do not fullfill the point.

1.14 Output Format

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

Ther is no displayed output in the whole class.

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

The error messages concerning of throwing exceptions, and they are comprehensive.

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

Ther is no displayed output in the whole class.

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)

Fullfilled point.

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

Fullfilled point.

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

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

Line 1458 could be factorized as:

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

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

There are no divisions in the assigned method.

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

Fullfilled point.

49. Check that the comparison and Boolean operators are correct

Fullfilled point.

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

Fullfilled point.

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

There are no implicit type conversions in the assigned method.

1.16 Exceptions

52. Check that the relevant exceptions are caught

Fullfulled point.

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

Fullfilled point.

1.17 Flow of Control

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

There are no switch statements.

55. Check that all switch statements have a default branch

There are no switch statements.

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

```
1458: for (int j = 0, size = orderByArray.size(); j <
    size; j++) {
1459:   ArrayList oa = (ArrayList) orderByArray.get(j);
1460:
1461:   if (constraint == null) {
1462:       constraint = new Constraint();
1463:   }
1464:
1465:   for (int k = 0, sizeK = oa.size(); k < sizeK; k
    ++) {
1466:       ConstraintFieldDesc ob = (
    ConstraintFieldDesc) oa.get(k);
1467:
1468:       if (ob.ordering < 0) {
1469:           constraint.addField(ob);
1470:           constraint.addOperation(ActionDesc.
    OP_ORDERBY_DESC);
1471:       } else {
1472:           constraint.addField(ob);
1473:           constraint.addOperation(ActionDesc.
    OP_ORDERBY);
1474:       }
1475:   }
1476: }
```

Lines 1458 and 1465 could be factorized as:

```
1458: for (int j = 0, j < orderByArray.size(); j++) {
1465:   for (int k = 0, k < oa.size(); k++) {
```

1.18 Files

57. Check that all files are properly declared and opened

There are no files in the whole class.

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

There are no files in the whole class.

59. Check that EOF conditions are detected and handled correctly

There are no EOF conditions in the whole class.

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

There are no file exceptions in the whole class.

Chapter 2

Other Problems

2.1 Initialization

1. Using *this* as parameter can be dangerous in the contructor because the object is not fully initialized.

```
195: public SelectQueryPlan(ActionDesc desc,
196:                        SQLStoreManager store,
197:                        Concurrency concurrency) {
[...]
211: retrieveDesc.setPlan(this);
```

2.2 Dodgy code

1. This cast is unchecked, and not all instances of the type casted from can be cast to the type it is being cast to.

```
940: if (((ConstraintOperation) nextNode).operation ==
    ActionDesc.OP_NOTNULL) {
```

2. Redundant check of a known non-null value against the constant null.

```
424: ArrayList group = config.getFetchGroup(groupID);
[...]
427: if (group != null) {
```

2.3 Performance

1. Use .isEmpty() instead of .size() == 0.

```
318: if (foreignFields.size() == 0) {
```

2. Method invokes inefficient Number constructor; use static valueOf instead

```
611: Object[] items = new Object[] {config.
   getPersistenceCapableClass().getName(),
612:                                     new Integer(
   statements.size())};
```

3. Unnecessary temporary when converting from String. Boxing types have parseXXX methods, which perform the conversion without creating the temporary instance.

```
159: private static final boolean MULTILEVEL_PREFETCH =
    Boolean.valueOf(
160:     System.getProperty(MULTILEVEL_PREFETCH_PROPERTY
   , "false")).booleanValue();
```

Appendix A

Document Information

A.1 Effort

Approximately **15 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