POLITECNICO DI MILANO

Computer Science and Engineering

Project of Software Engineering 2: "*myTaxiService*"

Code Inspection Document

Author: Andrea Maioli (mat. 852429)
Reference Professor: Mirandola Raffaela

Summary

1. Assigned Class and Methods

The assigned Class is ***VariableTable*** and is located in the *com.sun.jdo.spi.persistence.support.sqlstore.query.jqlc package*.

The assigned methods to inspect of this class are:

- public void ***markConstraint***(JQLAST *variable*, JQLAST *expr*)
- public void ***merge***(VariableTable *other*)
- protected void ***checkConstraint***(String *variable*, VarInfo *info*)

2. Functional Role

The ***VariableTable*** class represents a table containing information about specified variables.
For each variable there is an associated ***VarInfo*** object which contains:

- The status of the variable (*UNCHECKED*, *IN_PROGRESS*, *CHECKED*), used by checkConstraint() in order to avoid cyclic dependencies.
- The constraint of the variable
- The dependency information
- The usage information, that contains the list of the variables accessed from the VarInfo associated variable.

The aim of this class is to declare variables, define constraints on them and specify if the variable is used.
The last method to be called probably is checkConstraints() that will verify the constraints of each variable recursively. This method will also modify the Abstract Syntax Tree (AST) used by the query compiler, expanding each node with a subtree generated from the AST itself, by considering the constraint of the variable as the root node.

The inspected methods are:

- ***markConstraint***(JQLAST *variable*, JQLAST *expr*):
  This method modify the stored information of *variable*, setting it as a constraint with the given expression *expr*.
- ***merge***(VariableTable *other*):
  This method merge the VariableTable *other* into the current object.
  During its execution, this method verify if a variable is present in both the variable tables, and if it has been set as a constraint in both of them, the constraint must be the same for each variable table else an error is return. If the variable is set as a constraint in only one of the variable tables, the constraint is not considered and is removed from the variable.
- ***checkConstraint***(String *variable*, VarInfo *info*):
  This method will check the stored information of a variable, verifying that there are no cyclic dependencies and if the variable has a constraint set, it has also the used parameter not empty.
  At the end of this method, there is a call to attachConstraintToUsedAST() that will attach to the node of the variables stored in the VarInfo.usedHashSet, the subtree generated from the AST, by considering the constraint as root node in the AST itself, if the node has no child

3. List of Issue found applying the checklist

3.1. Naming Conventions

- ***markConstraint***:
  - Line 197: variable "*name*" can be named as "*variableName*", it could be more meaningful.
  - Line 198: variable "*entry*" can be named as "*info*" for consistency (in all the other methods of the analyzed class, all the VarInfo object are stored in a variable called "*info*" or containing this word).
  - Line 208: variable "*old*" can be named as "*oldConstraintText*" in order to be more meaningful.
- ***merge***:
  - Line 221: variable "*name*" can be named as "*variableName*", it could be more meaningful.
- ***checkConstraint***:
  - No problem found for the Naming Conventions.

3.2. Indention

- ***markConstraint***:
  - Line 201-203: the indentation level is increased with the parentheses level, so they must be indented with 8 spaces instead of 4.
  - Line 209: the line is indented with five group of four spaces and two spaces. This can be seen as a deep indent due the align of the variable "*name*" with the variable "*messages*" present on the previous line, but there is an inconsistent usage of this indentation method, because the line 208 doesn't align with the first parameter of the called method on line 207 (which uses a standard indent).
- ***merge***:
  - No problem found for the Indentation checklist.
- ***checkConstraint***:
  - Line 291 and 307 uses a deep indentation and is consistent with the indentation used inside the method, but not with the indentation used inside all the other methods.

3.3. Braces

- ***markConstraint***:
  - Line 199: the if statement has only one statement to be executed and is not surrounded by curly braces.
- ***marge***:
  - No problem found for the Braces checklist.
- ***checkConstraint***:
  - No problem found for the Braces checklist.

3.4. File Organization

- No problem found for the File Organization checklist.

3.5. Wrapping Lines

- No problem found for the Wrapping Lines checklist.

3.6. Comments

- The ***VariableTable*** class is not sufficiently commented, there is no explanation of what the class is used for.
- ***markConstraint***:
  - The blocks of the code are not commented.
- ***merge***:
  - No problem found for the Comments checklist.
- ***checkConstraint***:
  - The blocks of the code can be commented more adequately, since only the switch-case statement is commented.

3.7. Java Source Files

- ***markConstraint***:
  - The JavaDoc of this method tells "The method sets the constraint filed of the ***VarInfo*** object to true." but actually the method sets the constraint filed equals to the "*expr*" variable.
- JavaDoc is incomplete for the class and the methods inside.
- There is no documentation specified for the *checkConstraints*, *checkConstraint* and *attachConstraintToUsedAST* methods.

3.8. Package and Import Statements

- No problem found for the Package and Import Statements checklist.

3.9. Class and Interface Declarations

- For this implementation of the methods in the whole class, coupling is adequate but since all the methods accessing the **VarInfo** objects, directly access its variables it is better to implement getters and setters for the **VarInfo** class.

3.10. Initialization and Declarations

- ***markConstraint***:
  - Line 203: the declaration of the variable "*old*" is not at the beginning of a block. It could be declared at the begin of the block and initialized after the verification of the variable "*entry*" (the variable "*old*" value is dependent upon the value of the variable "*entry*").
- ***merge***:
  - No problem found for the Initialization and Declarations.
- ***checkConstraint***:
  - No problem found for the Initialization and Declarations.

3.11. Method Calls

- No problem found for Method Calls checklist.

3.12. Arrays

- No problem found for Arrays checklist.

3.13. Object Comparison

- No problem found for Object Comparison. (the only comparisons without the method "*.equals*()" is made to check if an object exists using the *obj == null* condition)

3.14. Output Format

- No problem found for Output Format checklist.

3.15. Computation, Comparisons and Assignments

- Both *merge* and *checkConstraint* methods refers to a variable defined in the VarInfo class, which defines a finite set of named constants that can be found on line 99-101. Those constants can be set in an Enum object.

3.16. Exceptions

- No problem found for Exceptions. There are methods that throws unchecked exception (such as *JDOFatalInternalException* and *JDOUnsupportedOptionException*) which are thrown only if the methods are not called properly (e.g.: the variable not exists or there is a duplicate variable). These exceptions are not mandatory to be declared as thrown, but some classes declare them (like ErrorMsg) and also there is no mention in the JavaDoc of the exceptions.

3.17. Flow of Control

- ***markConstraint***:
  - No problem found for Flow of Control.
- ***merge***:
  - No problem found for Flow of Control.
- ***checkConstraint***:
  - Line 281: the switch has no default branch.
  - Line 289: this case is not addressed by a "return" or "break", but there is an exception that block the execution of consecutive cases.

3.18. Files

- No problem found for Files checklist.

4. Other problems

- Due to the low documentation and the imprecisions in the JavaDoc, it was very difficult to properly understand the possible usage of the **VariableTable** class.
- **markConstraint**:
    - Line 204: in order to be uniform to the style used in each comparation, entry.constraint==null can be wrote as “entry.constraint == null” by putting a space after and before the “==”.

5. Reference Documents

- “Assignment 3: Code Inspection” pdf document
- Web References for code conventions:
    - Oracle Code Conventions: http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-136091.html
    - Jalopy: http://jalopy.sourceforge.net/existing/indentation.html