CODE INSPECTION

ANDREA MAIOLI

Assigned Class

- ***VariableTable*** class represents a table containing information about previously specified variables.
- For each variable there is an associated ***VarInfo*** object which contains:
  - Status of the variable (UNCHECKED, IN_PROGRESS, CHECKED), used by checkConstraint() in order to avoid cyclic dependencies.
  - Constraint (if any)
  - Dependency information: if the variable depends on another one
  - [***PROBLEM***] Usage information: which variable uses the current one or which variable is used by the current one? Is difficult to understand due to the imprecise documentation provided. I'm supposing that this HashSet will contain the variables accessed by the associated object.
- 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 recursively call checkConstraint() on each variable.

Assigned Methods

- ***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.****used* HashSet, the subtree generated from the AST, by considering the constraint as root node in the AST itself, if the node has no child.
    - The AST is an Abstract Syntax Tree used by the query compiler.

Assigned Methods

- ***markConstraint***(JQLAST *variable*, JQLAST *expr*):
  - This method modify the stored information of a variable, setting its saved constarint equals to *expr*.

- ***merge***(VariableTable *other*):
  - This method merge the VariableTable object *other* into the current object.
  - If a variable is present into both the objects:
    - If in one of two objects there is no constraint set, the constraint is set to null.
    - If in both the objects there is a constraint set, the constraint must be the same. An exception in thrown otherwise.

Naming Convetions Issues

- ***markConstraint(JQLAST variable, JQLAST expr)***:
  - Line 197: variable "*name*" can be named as "*variableName*", it could be more meaningful.

```
197         String name = variable.getText();
```

  - 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).

```
198         VarInfo entry = (VarInfo)varInfos.get(name);
```

  - Line 208: variable "*old*" can be named as "*oldConstraintText*" in order to be more meaningful.

```
204         String old = (entry.constraint==null ? null : entry.constraint.getText());
```

Naming Convetions Issues

- ***merge(VariableTable other)***:
  - Line 221: variable "*name*" can be named as "*variableName*", it could be more meaningful

```
221                 String name = (String)i.next();
```

Indention Issues

- ***markConstraint(JQLAST variable, JQLAST expr)***:
  - Line 201-203: the indentation level is increased with the parentheses level, so they must be indented with 8 spaces instead of 4.

```
200                 throw new JDOFatalInternalException(I18NHelper.getMessage(
201                     messages,
202                     "jqlc.variabletable.markconstraint.varnotfound", //NOI18N
203                     name));
```

  - 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).

```
207                 errorMsg.unsupported(variable.getLine(), variable.getColumn(),
208                     I18NHelper.getMessage(messages, "jqlc.variabletable.markconstraint.multiple", //NOI18N
209                                           name));
```

Indention Issues

- **checkConstraint(String variable, VarInfo info)**:
  - 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.

```
290                 I18NHelper.getMessage(messages, "jqlc.variabletable.checkconstraint.cycle", // NOI18N
291                                       variable));
```

```
305               throw new JDOUnsupportedOptionException(
306                   I18NHelper.getMessage(messages, "jqlc.variabletable.checkconstraint.unused", //NOI18N
307                                         variable));
```

Braces Issues

- ***markConstraint(JQLAST variable, JQLAST expr)***:
  - Line 199: the if statement has only one statement to be executed and is not surrounded by curly braces.

```
199             if (entry == null)
200                 throw new JDOFatalInternalException(I18NHelper.getMessage(
201                     messages,
202                     "jqlc.variabletable.markconstraint.varnotfound", //NOI18N
203                     name));
```

Comments Issues

- The ***VariableTable*** class is not sufficiently commented, there is no explanation of what the class is used for.

- ***markConstraint(JQLAST variable, JQLAST expr)***:
  - The blocks of the code are not commented.

- **checkConstraint(String variable, VarInfo info)**:
  - The blocks of the code can be commented more adequately, since only the switch-case statement is properly commented.

Java Source Files Issues

- ***markConstraint(JQLAST variable, JQLAST expr)***:
  - The JavaDoc of this method is: "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.

- There is no JavaDoc specified for this methods:
  - checkConstraints
  - checkConstraint
  - attachConstraintToUsedAST

- Javadoc is incomplete for the whole class and the methods inside.

Class and Interface Declarations Issues

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

Initialization and Declarations Issues

- ***markConstraint(JQLAST variable, JQLAST expr)***:
  - Line 204: 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*").

```
195     public void markConstraint(JQLAST variable, JQLAST expr)
196     {
197         String name = variable.getText();
198         VarInfo entry = (VarInfo)varInfos.get(name);
199         if (entry == null)
200             throw new JDOFatalInternalException(I18NHelper.getMessage(
201                 messages,
202                 "jqlc.variabletable.markconstraint.varnotfound", //NOI18N
203                 name));
204         String old = (entry.constraint==null ? null : entry.constraint.getText());
```

Computation, Comparisons and Assignments Issues

- 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.

```
 99          static final int UNCHECKED = 0;
100          static final int IN_PROGRESS = 1;
101          static final int CHECKED = 2;
```

Exceptions Issues

- 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 duplicate).
- There is no problem for the Exceptions Checklist, and the above exceptions are not mandatory to be declared as thrown, but some classes of *GlassFish* declare them as thrown (like **ErrorMsg**). There is no mention in the JavaDoc of the exceptions.

Flow of Control Issues

- **checkConstraint(String variable, VarInfo info)**:
  - 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.

```
281         switch (info.status)
282         {
283         case VarInfo.UNCHECKED:
284             // if unchecked, start checking
285             info.status = VarInfo.IN_PROGRESS;
286             break;
287         case VarInfo.IN_PROGRESS:
288             // if this VarInfo is currently processed we have a cyclic dependency
289             throw new JDOUnsupportedOptionException(
290                 I18NHelper.getMessage(messages, "jqlc.variabletable.checkconstraint.cycle", // NOI18N
291                                       variable));
292         case VarInfo.CHECKED:
293             // if alreday checked just return
294             return;
295         }
```

Other Problems

- Due to the low documentation and the imprecisions in the JavaDoc, I found very difficult to properly understand the possible usage of the **VariableTable** class.

- ***markConstraint(JQLAST variable, JQLAST expr)***:
  - 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 “==”.

```
204         String old = (entry.constraint==null ? null : entry.constraint.getText());
```

References

- CheckList
- Oracle Code Conventions: http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-136091.html
- Jalopy:
  - http://jalopy.sourceforge.net/existing/indentation.html

Thank You

ANDREA MAIOLI