Software Engineering 2 Code Inspection Project

Gabriele Giossi (Id: 854216)

v1.0 - 04/01/2016

Contents

1 Inspection Subject

The subject of this inspection assignment was the source code for the Glassfish application server; This group has been assigned the following code:

- assertPKUpdate(FieldDesc f, Object value);
- loadForRead();
- loadForUpdate();

These three methods are all included inside the SQLStateManager.java source file, located at the following path inside the provided source code root folder:

appserver/persistence/cmp/support-sqlstore/src/main/java/com/sun/jdo/spi/persistence/support/sqlstore/SQLStateManager.java

The inspection was performed by running the code through a checklist (available on the assignment document, which will have to be referred to for details, it will be provided on the same repository on which this document will be released), in order to identify any possible issues.

2 Functional role of code

The three assigned methods are all relevant to the class SQLStateManager, which is pretty self-explicative already in its name: the class is deputed to effectively interact with SQL objects and manage their state; going in details with the assigned methods to inspect, the following has been understood:

1. Method assertPKUpdate(FieldDesc f, Object value) is a more complex way of comparing values, probably used extensively in other sections of the source code; it very simply checks that objects are conformed to an expected value and, if not, throws an exception;
2. Method loadForRead(), citing the javadoc: "Triggers the state transition for READ and registers the instance in the transaction cache." In short, it is a method that triggers a read-lock on a resource and makes it so that all transactions and the scheduler know that there is a read-type lock active. The method also incorporates, in case of errors or faults, rollback procedures to restore the old states, in case of any problem, in order to maintain atomicity. It is important to note that the method contains a fail-safe mechanism that prevents multiple threads each calling loadForRead() to place multiple locks on a given resource;
3. Method loadForUpdate() is very similar to the previous method loadForRead(), in that it attempts locking a resource and registers something in the transaction cache; similarly to the other method, only a single thread can execute loadForUpdate on a given instance (as evidenced by comment lines 4208-4209) - thus it is far more stringent, but necessary, as the method also prepares DFG fields for manipulation, in this case, an update; the strong constraint of having only one thread able to execute loadForUpdate on a given instance is necessary to preserve the fundamental properties of transactional systems;

3 Issues discovered related to checklist

Based on the checklist sections:

- **Naming Conventions**: method "assertPKUpdate" assigns a one-character name to one of its parameters (line 4093); thus, point 2 of the checklist is not fully covered;

```
4093    private void assertPKUpdate(FieldDesc f, Object value) {
        ...
        }
```

- **Indention**: no issues, indention is done correctly and consistently;
- **Braces**: no issues; the bracing style used is the "Kernighan and Ritchie" style;
- **File Organization**: no issues, the file is organized and contains only a single class;
- **Wrapping Lines:** no issues, line breaks are not used nor needed;
- **Comments**: no issues;
- **Java Source Files**: javadoc for method assertPKUpdate is not showing any description; simple as the method might be, it is necessary to have some description in the javadoc:
- **Package and Import Statements**: no issues, statements are correctly ordered and written;
- **Class and Interface declarations**: class variables for the SQLStateManager class are in mixed order, with a public variable declared below protected and private variables: the code w.r.t. point 25.D is not compliant (lines following are lines 150-163 of the source file);

```
/** I18N message handler. */
private final static ResourceBundle messages = I18NHelper
    .loadBundle(
            SQLStateManager.class);

/** Name of the USE_BATCH property. */
public static final String USE_BATCH_PROPERTY =
       "com.sun.jdo.spi.persistence.support.sqlstore.
           USE_BATCH"; // NOI18N

/**
 * Property to swich on/off batching. Note, the default
    is true, meaning we
 * try to do batching if the property is not specified.
 */
```

```
private static final boolean USE_BATCH = Boolean.valueOf
    (
System.getProperty(USE_BATCH_PROPERTY, "true")).
    booleanValue(); // NOI18N
```

- **Initialization and Declarations**: no issues;
- **Method Calls**: no issues:
- **Arrays**: point 39 for method loadForUpdate shows criticity at line 4216: an ArrayList is declared, but no constructor is called;

```
4214    persistenceManager.setFlags(persistentObject, READ_WRITE_OK);

4216    ArrayList fields = persistenceConfig.fields;

4218    try {
            ...}
```

- **Object Comparison**: All comparations are done through "==" or "!=" instructions; being comparations of integer type data, it can be accepted; there are no object comparisons;
- **Output Format**: on point 42 of the checklist, sudden termination of the two methods loadForread and loadForUpdate (lines 4158-4159 and lines 4210-4211) provides a simple return command, without any sort of exception handling or other notification of termination;

```
4158        if (oldFlags != LOAD_REQUIRED) {
            return;
            }

4210        if (oldFlags == READ_WRITE_OK) {
            return;
            }
```

- **Computation, Comparisons and Assignments**: no issues;
- **Exceptions**: no issues;
- **Flow of Control**: no issues due to absence of switch statements, and loops are correctly executed;
- **Files**: no file interaction is used in the methods;

4 Other highlighted problems

Given the assigned code fragment for inspection, the possible issues that could arise are very few, in part owing to the rather simple construction and working of the methods; there are, however, some contexts that might cause the code to be unfriendly in reporting errors or notifying limit-case events; this could be refined:

1. Method assertPKUpdate could only be called by the class itself, being private; it is used only in two other methods, not object of this group's inspection assignment; given the simple nature of assertPKUpdate's exception handling, however, the possible issue that is raised is how said exception is handled in case it is thrown, and whether or not it is notified to the user, of if it is important to do so;

2. It has been already noted in the checklist-related issues about the sudden, unexpected, and non-notified return statement of methods loadForRead and loadForUpdate - given that it is a simple return statement, without exception throwing or notification to the end user, there could possibly be a problem where the termination of a method call done this way leaves the user completely unaware of the fact that the triggering the methods are responsible for has not been carried out;

In general though, the methods that have been inspected are effectively simple enough to warrant that many possible limit-cases that could compromise the functionality of more complex code here are not much of a problem. All the reported possible issues here are noted with the disclaimer that, being a simple code inspection, there was no active debugging at runtime, nor analysis of all pieces of code that interact with the assigned methods; both these passages can rule out any of the possible issues reported here;

5 Reference

5.1 Hours of working

Towards the redaction of the document in fulfilment of the deadline and the inspection of the assigned code, the group members worked the following number of hours:

Gabriele Giossi: 15 hours.

5.2 Used tools

Code analysis was performed using the Eclipse Java EE tool, used also to generate and inspect the javadoc for the assigned sections of code.