myTaxiService

Code Inspection

A.Y. 2015/2016

Politecnico di Milano

Version 1.0

Cattaneo Michela Gaia, matr. 791685
Barlocco Mattia, matr. 792735

January 5th, 2015

Contents

1 Introduction

1.1 Purpose

This document represents the Code Inspection document, whose main goal is to examine source code in order to discover bugs, verifying that coding conventions are respected and look for potential issues. This part is necessary to improve the quality of the code and to assure that there are no errors in it.
This document is addressed to the project development teams, who need to work on code without mistakes before the production phase, in order to avoid future possible problems.

1.2 Document structure

This document is divided in five main parts:

- **Introduction:** this section describes the document in general and its purpose.
- **Classes assigned:** this section specifies the name of the classes and methods assigned to the group.
- **Functional roles:** this section shows the role of the class and of each method assigned and their functionalities.
- **Issues:** this section verifies that all the points of the code inspection checklist are respected in the classes and methods assigned.
- **Other problems:** this section deals with other possible issues found in the code, referring to the lines generating a bug and how it can create problems in the program.
- **References**: this section lists all the useful materials necessary for this document.

2 Classes And Methods Assigned

2.1 Package

com.sun.jdo.spi.persistence.support.sqlstore

2.2 Class

SQLStateManager

2.3 Methods

- addCollectionRelationship
- setForeignKey
- addJoinTableEntry
- prepareUpdateFieldSpecial

3 Functional Roles

These methods are part of the SQLStateManager, the class that manages the state transitions and the contents of the fields, taking care of adding, removing and updating the values and the operations in the tables and the relationships between the fields.

3.1 Method 1: addCollectionRelationship

This method sets the relationship for the objects added to a collection relationship. Given a list of object to add to the relationship and a list of newly registered state managers, this function transforms and adds all the elements of the **addedList** to the **newlyRegisteredSMs** list, by checking that the elements are not null and making them autopersistent. It also checks that the element added has not been deleted and updates the relationship in the data store.
This method is used when it is necessary to apply the updates in the data store, in particular it processes the updates in the collections by adding the relationships between them.
The evidence of its functionality can be found in the JavaDoc and checking the function calls in the SQLStateManager class.

3.2 Method 2: setForeignKey

This method is used to set the foreign key corresponding to the relationship field.
The **fieldDesc** and **inverseFieldDesc** parameters are used for the iteration on the local or foreign fields. This function calls its homonym **setForeignKey**, a method with other parameters, that actually sets the local field corresponding to a foreign key column to the new value for the relationship update.
This method is called in the **processForeignKey** method in order to set the foreign key on the added object. It is useful when an update of a relationships in the data store is needed.
The evidence of its functionality can be found in the JavaDoc and checking the function calls in the SQLStateManager class.

3.3 Method 3: addJoinTableEntry

This method is used to schedule the creation of a join table entry between this and the added State Manager, after removing every scheduled removal. Note that the side creating the join table entry has to wait for the other to become persistent and the situation in which the field descriptor passed is null needs to be handled.
It is needed in order to process the join table entries, in particular to schedule the join table entry to the added object. It is part of the update of the relationships in the data store.
The evidence of its functionality can be found in the JavaDoc and checking the function calls in the SQLStateManager class.

3.4 Method 4: prepareUpdateFieldSpecial

This is a different version of the **prepareUpdateField** method. They both are very important for the recording of changes to fields, as they prepares the field described by the parameter **fieldDesc** for update. In fact this method is called in different other methods, with various functionalities: when a relationship is removed or added and when it is necessary to update the values for fields tracking field.
The evidence of its functionality can be found in the JavaDoc and checking the function calls in the SQLStateManager class.

4 Issues

These are the list of the lines of code with the issues found applying the checklist.

4.1 Class

- **line 75:** There is no JavaDoc for the class.
- **line 77-162:** these lines does not respect the point 25.D and 25.E of the checklist, the public static or instance variables are not declared before the private ones.
- **line 151:**

```
private final static ResourceBundle messages = I18NHelper.loadBundle(
                                              SQLStateManager.class);
```

  Here the ResourceBundle is declared as a constant and it is not capitalized, as it is said in the naming conventions at point 7 of the checklist.

- **line 219:**

```
private void registerInstance(boolean throwDuplicateException,
                 ArrayList newlyRegisteredSMs, LifeCycleState oldstate);
```

  There is no JavaDoc for this method.

- **line 270:**

```
private void unsetMaskBit(int index, int mask);
```

  There is no JavaDoc for this method.

- **line 283:**

```
private void clearMask(int mask);
```

  There is no JavaDoc for this method.

- **line 308:**

```
private void newFieldMasks();
```

  There is no JavaDoc for this method.

- **line 400:**

```
public synchronized void replaceObjectField(String fieldName, Object o);
```

  There is no JavaDoc for this method.

- **line 408:**

```
public synchronized void makeDirty(String fieldName);
```

  There is no JavaDoc for this method.

- **line 508:**

```
public void makePresent(String fieldName, Object value);
```

  There is no JavaDoc for this method.

- **line 560:**

```
private void makeAutoPersistent(Object pc);
```

  There is no JavaDoc for this method.

- **line 1108:**

```
private boolean compareUpdatedFields(SQLStateManager next);
```

There is no JavaDoc for this method.

- **line 1542:**

```
private void reset(boolean retainValues, boolean wasNew, boolean keepState);
```

There is no JavaDoc for this method.

- **line 1706:**

```
private void markKeyFieldsPresent();
```

There is no JavaDoc for this method.

- **line 2154:**

```
private SQLStateManager copyPersistent();
```

There is no JavaDoc for this method.

- **line 2240:**

```
private Object cloneObjectMaybe(Object source);
```

There is no JavaDoc for this method.

- **line 4087:**

```
private void assertNotPK(int fieldNumber);
```

There is no JavaDoc for this method.

- **line 4093:**

```
private void assertPKUpdate(FieldDesc f, Object value);
```

There is no JavaDoc for this method.

4.2 Method 1: addCollectionRelationship

- **line 3384:**

```
SQLStateManager addedSM = getAddedSM(addedObject, newlyRegisteredSMs);
```

This line exceeds the 80 characters suggested at point 13 of the checklist and it is possible to reduce the number of characters in this line, by wrapping after the comma in the **getAddedSM** call, as said at point 15 of the checklist.

- **line 3409:**

```
updateRelationshipInDataStore(fieldDesc, addedSM, null, inverseFieldDesc,
false);
```

This line exceeds the 80 characters suggested at point 13 of the checklist and it is possible to reduce the number of characters in this line, by wrapping after the commas that separates the parameters of the method, as said at point 15 of the checklist.

4.3 Method 2: setForeignKey

- **line 3451-3452:**

```
LocalFieldDesc la = (LocalFieldDesc) fieldDesc.localFields.get(i);
LocalFieldDesc fa = (LocalFieldDesc) fieldDesc.foreignFields.get(i);
```

This lines exceed the 80 characters suggested at point 13 of the checklist and it is possible to reduce the number of characters in this lines, by wrapping after the equal operator after **la** and **fa**, as said at point 15 of the checklist.

- **line 3458-3459:**

```
LocalFieldDesc la = (LocalFieldDesc) inverseFieldDesc.foreignFields.get(i);
LocalFieldDesc fa = (LocalFieldDesc) inverseFieldDesc.localFields.get(i);
```

This lines exceed the 80 characters suggested at point 13 of the checklist and it is possible to reduce the number of characters in this lines, by wrapping after the equal operator after **la** and **fa**, as said at point 15 of the checklist.

4.4 Method 3: addJoinTableEntry

- **line 3509:**

```
* RESOLVE: What happens, if a field descriptor is null, e.g. for one
* way relationships, as the descriptors are taken as keys during scheduling?
```

This comment can be deleted as long as it is no longer needed, this is handled by the exception thrown at line 3542.

- **line 3526:**

```
if (fieldDesc != null && getUpdateDesc().removeUpdatedJoinTableRelationship(
fieldDesc, addedSM, ActionDesc.LOG_DESTROY) == false)
```

This line does not agree with point 44 of the checklist: in fact the second expression of the **if** returns a boolean value and it is not necessary to compare it with **false**. Denying the expression, adding the pertinent parenthesis, is a better solution. It does not respect the point 13 and 15, because it is better to wrap the line after the AND operator and after the commas that separate the parameters, in order to reduce the characters.

- **line 3530:**

```
if (inverseFieldDesc == null || addedSM.getUpdateDesc().
        removeUpdatedJoinTableRelationship(
                inverseFieldDesc, this, ActionDesc.LOG_DESTROY) == false)
```

This line does not agree with point 44 of the checklist: in fact the second expression of the **if** returns a boolean value and it is not necessary to compare it with **false**. Denying the expression, adding the pertinent parenthesis, is a better solution. It does not respect the point 13 and 15, because it is better to wrap the line after the OR operator and after the commas that separate the parameters, in order to reduce the characters.

- **line 3535:**

```
getUpdateDesc().recordUpdatedJoinTableRelationship(
```

This line does not respect point 15 of the checklist where it is said that line breaks occurs after a comma or an operator. Wrapping this line after the comma that separates the parameters is a better solution, that still respects point 13.

- **line 3664:**

```
//should throw an exception
```

According to point 52, here it is necessary to handle the exception, as suggested in the comment.

4.5 Method 4: prepareUpdateFieldSpecial

- **line 3626:**

```
logger.fine("sqlstore.sqlstatemanager.prepareupdatefieldspl", items);
```

This line exceeds the 80 characters suggested at point 13 of the checklist and it is possible to reduce the number of characters in this line, by wrapping after the commas that separates the parameters of the method, as said at point 15 of the checklist.

- **line 3673:**

```
SCOCollection c = (SCOCollection) persistenceManager.
                                  newCollectionInstanceInternal(
```

This line exceeds the 80 characters suggested at point 13 of the checklist and it is possible to reduce the number of characters in this line, by wrapping after the equal operator and bring on the same line the first parameter of the method, as said at point 15 of the checklist. Doing so, it does not exceed the 120 characters suggested at point 14.

- **line 3622:**

```
boolean debug = logger.isLoggable();
```

This variable should be declared at the beginning of the block, which is at the beginning of the function, as said at point 33 of the checklist.

- **line 3629-3631:**

```
boolean optimistic = persistenceManager.isOptimisticTransaction();
boolean xactActive = persistenceManager.isActiveTransaction();
boolean nontransactionalRead = persistenceManager.isNontransactionalRead();
```

These variables should be declared at the beginning of the block, which is at the beginning of the function, as said at point 33 of the checklist.

- **line 3641:**

```
LifeCycleState oldstate = state;
```

This variable should be declared at the beginning of the block, which is at the beginning of the function, as said at point 33 of the checklist.

- **line 3668:**

```
Object value = fieldDesc.getValue(this);
```

This variable should be declared at the beginning of the *else* block, as said at point 33 of the checklist.

5 Other Problems

No other problems have been highlighted.

6 References

- Code inspection assignment (`https://github.com/AntoniniP/CodeInspectionChecklist/blob/master/original.pdf`)
- Checklist (`https://github.com/AntoniniP/CodeInspectionChecklist/blob/master/document.pdf`)
- Code documentation (`glassfish.pompel.me`)
- Source code (`https://svn.java.net/svn/glassfish~svn/tags/4.1.1`)