Politecnico di Milano

Computer Science and Engineering

Project of Software Engineering 2

Code Inspection

Authors:

Antonio Iannacci - 854157

Daniele Romanini - 854732

Federico Seri - 854032

Reference Professor: Mirandola Raffaela

TABLE OF CONTENT

1. Assigned Classes and methods

For this project we had to perform the inspection of some methods.

To our group was assigned the class: **EntityContainer.java** from the source code of "*Glassfish 4.1 application server*".

The full path of the class is:
appserver/persistence/entitybean-container/src/main/java/org/glassfish/persistence/ejb/entitybean/container/EntityContainer.java

From this class 5 methods were chosen:

1. **releaseContext**(**EjbInvocation** inv) - Start Line: 693
2. **postCreate**(**EjbInvocation** inv , **Object** primaryKey) - Start Line: 815
3. **postFind**(**EjbInvocation** inv , **Object** primaryKeys , **Object [ ]** findParams) - Start Line: 926
4. **getEJBLocalObjectForPrimaryKey**(**Object** pkey , **EJBContext** ctx) - Start Line: 1034
5. **removeBean**(**EJBLocalRemoteObject** ejbo , **Method** removeMethod , **boolean** local) - Start Line: 1097

1.1. Glossary

Following, we report some definition frequently used in the document.

- **EJB:** Enterprise Java Beans. EJB is a server-side software component that encapsulates the business logic of an application.
- **CONTEXT:** A collection of data, used for maintaining the state of the system.
- **TRANSACTION:** A Transaction object is created corresponding to each global transaction creation. The Transaction object can be used for resource enlistment, synchronization registration, transaction completion, and status query operations.
  (ref. :
  https://docs.oracle.com/javaee/6/api/javax/transaction/Transaction.html)
- **OBJECT:** Object may be either local or remote. If it is a local object (that is, running in the same VM as the client), invocations may be directly serviced by the object instance, and the object reference could point to the actual instance of the object implementation class. If a CORBA object is a remote object (that is, running in a different VM from the client), the object reference points to a stub (proxy) which uses the ORB machinery to make a remote invocation on the server where the object implementation resides.
  (Object. Ref:
  http://docs.oracle.com/javase/7/docs/api/org/omg/CORBA/Object.html)
  (LocalObject. Ref.:
  http://docs.oracle.com/javase/7/docs/api/org/omg/CORBA/LocalObject.html
)
- **TX:** Abbreviation for "Transaction"
- **ACTIVETXCACHE:** All INCOMPLETE_TX bean instances (Instances of bean associated to incomplete transactions) are stored in the ActiveTxCache. Instances in INVOKING state which have transactions associated with them are also in ActiveTxCache.
  (ref.:
  http://glassfish.pompel.me/org/glassfish/persistence/ejb/entitybean/container/EntityContainer.html)
- **CASCADE DELETE:** If the parent entity is removed from the current persistence context, the related entity will also be removed.
  For example, a line item is part of an order; if the order is deleted, the line item also should be deleted. This is called a cascade delete relationship.
  (ref.: https://docs.oracle.com/cd/E19798-01/821-1841/bnbqm/index.html)
- **RE-ENTRANT CALL:** a called to a subroutine is named reentrant if it can be interrupted in the middle of its execution and then safely called again ("re-entered") before its previous invocations complete execution.
  (ref.: https://en.wikipedia.org/wiki/Reentrancy_(computing)).

1.2. General function of the class

We have analysed the general functional role of the class and then we analysed each method.

From the Javadoc we can see the role of the class:
"The EntityContainer manages collections of EJBs in different states. The 5 states of an EntityBean (an EJB can be in only 1 state at a time):"

- **POOLED**
- **READY**
- **INVOKING**
- **INCOMPLETE_TX**
- **DESTROYED**

Link to the Javadoc:
http://glassfish.pompel.me/org/glassfish/persistence/ejb/entitybean/container/EntityContainer.html

1.3. Functional Role – Method 1

The method releaseContext is used to manage and release the invocation's context. It is inherited from the class BaseContainer in which the method releaseContext is declared as abstract and protected; this means that all the classes that extend BaseContainer, included EntityContainer, can implement this method.

The method has the following parameters:

- **EjbInvocation inv:** the object containing the state associated with the invocation of the EJB.

To perform its work, the method needs to define some variables:

- context: it is referred to the context of the specific invocation (passed as parameter);
- decrementedCalls: it is used to manage the call related to a context; every time that a context invokes the method decrementCalls, the variable is set TRUE;
- status: it is a variable used to store the last transaction's status from the context through the function getLastTransactionStatus.

As written in the javadocs and the comments, releaseContext is called from BaseContainer.postInvoke after EntityContainer.preInvokeTx has been called.

Line 697:

- In case the context does not exist (thus, is in DESTROYED state), the method returns;

Lines from 700: there is a try-catch-finally block.

TRY BLOCK (Lines from 700 to 784): there are 5 different mutual exclusive blocks that could be executed (the condition to execute the block are checked in order: if the first one is executed, no other blocks can be executed, ecc.)

Case 1) *Lines from 701 to 718:* the context has some re-entrant calls associated.If the Ejb associated to the invocation has been removed, it is removed from the table of incomplete transactions, in order to prevent further re-entrant calls (the Ejb does not exist anymore: it does not make sense to make a call on it). Otherwise, if the context considered is in INVOKING state, the state contained in the invocation object is flushed). This is done because an EJB in INVOKING state is not stored anywhere (as written in the Javadoc of EntityContainer class).

Case 2) *Lines from 718 to 740:* if the object does not exists neither locally nor remotely:
The number of calls associated to the context is decremented.

- If the invocation considered is a "creation invocation" (in fact, as written in the comments, this block can be executed when an exception is thrown during a creation process), the transaction of the context is set to null and

the context is added to the POOLED Ejbs (see the possible states of EntityContainer class).
- Otherwise, if the transaction does not exist, the Ejb is added to the POOLED ones.
- Otherwise, the state of the transaction associated with the context is marked as incomplete (INCOMPLETE_TX), thus it is ready for invocations, but the transaction is still in progress

Case 3) *Lines from 740 to 760*: if the object associated to the invocation has been removed, the removal of the EJB is completed deleting any transaction associated to the context (if there are any), then is unset the removed flag in case of the EJB ref is held by the client and used again.
After this the number of call associated to this context is decremented by 1 and if there are no transaction associated to the context the EJB, it is added to the Pooled EJB.
Otherwise the state is set to incomplete because the transaction is not done yet.

Case 4) *Lines from 760 to 777:* If no transaction is associated to the context, it gets the status number of the last transaction, decrements by 1 the number of call associated to this context and set the status of the last call to -1.
Then if the status was equals to: -1, committed or without active transaction, the context of the current EJB is added to the set of ready EJB else the EJB is clean and added to the Pooled EJB. (With clean I mean set ready to be added to the Pooled EJB)

Case 5) *Lines from 777 to 785*: this block is executed if no one of the previous has been executed.
The state of the transaction associated with the context is marked as incomplete (INCOMPLETE_TX), thus is ready for invocations, but the transaction is still in progress. The invocation object is flushed.

CATCH BLOCK (Lines from 785 to 789): When an exception is caught while executing the "try-block", the error is written in the log and a new EJBException is thrown.

FINALLY BLOCK (Executed anyway): if the number of calls in the context was not decremented previously, touch method is called (we are not able to specify what this method does, since there is not any comment provided in the Javadoc of the EntityContextImpl class).

1.4. Functional Role – Method 2

This method is the implementation of *postCreate* method specified in the interface Container, and overrides the *postCreate* method in the class *BaseContainer*, extended by the given *EntityContainer* class.
The *postCreate* method is the end part of a creation process of an EJB Object.
It aims to associate the created EJB Object with the context.
It will not be called if the *ejbCreate* method throws an exception: in fact, in this case, there will not be any object to manage (because of the failed creation).

The method has the following parameters:

- **EjbInvocation inv:** the object containing the state associated with the invocation of the EJB.
- **PrimaryKey primaryKey:** the value returned from ejbCreate (ref.: Javadoc of EntityContainer, in the description of the method. Link: http://glassfish.pompel.me/org/glassfish/persistence/ejb/entitybean/container/EntityContainer.html)

Lines from 818 to 820: if primaryKey returned from the ejbCreate method is null, there was a problem, so the method ends throwing an exception.

Lines from 822 to 838: do different things based on the fact that the view of Data is local or remote (ref: Javadoc of BaseContainer, class extended by EntityContainer.
Link: http://glassfish.pompel.me/com/sun/ejb/containers/BaseContainer.html).

CASE 1 (Lines from 822 to 828): the view of Data is remote and the invocation Object (inv, the parameter) is local (thus the invocation was through the 2.x (or earlier) Local client view, the 3.x local client view or a no-interface client view; ref.: Javadoc of isLocal method, in the class EjbInvocation. Link: http://glassfish.pompel.me/com/sun/ejb/EjbInvocation.html):

Create an actual ejb object (EjbObjectImpl, class implementing methods of EjbObject), referencing to the internal implementation of the ejb object (the object needed will be recognized by passing the primary key to the method internalGetEJBObjectImpl).
Moreover, the method associates the context of the invocation (contained in the EjbInvocation object passed as parameter) with the ejbObject just created.

CASE 2 (Lines from 830 to 838): the view of Data is local:

Create EJBLocalObject irrespective of local/remote EjbInvocation (as written in the comments), calling the internalGetEJBLocalObjectImpl method also in this case (actually here it must be an override of the method, because the parameters passed are two instead of three). Finally, the method associates the context of the invocation with the object just create as the if-clause before, but here the object associated is the local one (created at line 834).

Line 840: create a context object and give it the reference to the context of the invocation Object (inv) passed as parameter of this method.

Lines from 841 to 845: if a transaction Object associated with the context object just created at line 840 exists (is not null), do the following: add the context object to the INCOMPLETE_TX table that contains ejb objects in an incomplete status. It is necessary to manage synchronization. (The Object is actually not complete, because it has a transaction object associated).

Line 847: Finally, this line modifies eventually the state of the context (in fact, probably, if the state is already set to true, this line will not modify it).

1.5. Functional Role – Method 3

The method has the following parameters:

- **EjbInvocation inv**: It has the information about the object to collect, it tells us if the object is local or not.
- **Object primaryKeys**: The primary key or the list of primary keys.
- **Object[] findParams**: It is not used, it is keep only to not change old call to this method. Probably a previous implementation used the parameters.

The task of this method is to get the EJBean associated to the primaryKey.
The primaryKey can be of different type:

- Case Enumeration: It is an Enumeration of primary key and for each key the function get the related EJBean. At the end returns the list of found objects.
- Case Collection: It is the same of Enumeration but in this case the keys are in a Collection.
- Case Object: It is a single key so returns a single object.

The case "Enumeration" goes from line 931 to 949, the case "Collection" goes from line 950 to 968 and the case "Object" goes from 969 to 978.

The research of the related EJBean is done only if the primary key is not null, then if the EJB is local is used the function "*getEJBLocalObjectForPrimaryKey"* else is used the function *getEJBObjectStub*.

"*Enumeration*" and "*Collection*" are two standard Interface from *java.util*.
(Here we can see their Javadocs:

- https://docs.oracle.com/javase/7/docs/api/java/util/Enumeration.html
- https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)

1.6. Functional Role – Method 4

This method is specified in the interface Container, and Overrides the method in the super-class *BaseContainer*.
The aim is to obtain an Entity *EJBLocalObject* corresponding to the primary key, and it is used by the *Persistence Manager* (ref: Javadoc of *Container* interface. Link: http://glassfish.pompel.me/com/sun/ejb/Container.html).
In the comments and Javadoc it is possible to read various examples of the usage of this method, where it is written that it is called only during cascade delete, and it is a private API between *Persistence Manager* and *Container.*

Understanding this method in details has been difficult, because of a lack in the comment in Javadocs and the impossibility to find the documentation associated to some objects (see "Other problems" section).

The method has the following parameters:

- **PrimaryKey Pkey:** The primary key for which the EJBLocalObject is required
- **EntityContext Ctx:** The context associated with the bean from which the accessor method is invoked

The method returns the EJBLocalObject associated with the PrimaryKey or null if it is cascade deleted.

Lines from 1042 to 1060: if the performing cascade delete is before the deletion of the Ejb associated to the context, do the following: try to get the reference to the current performing transaction.
Then, create an *ActiveTxCache* object and assign it the null value (if there is not a current performing transaction) or get the reference of *the ActiveTxCache* object where all the existing incomplete transactions are stored.

Lines from 1051 to 1057: if activeTxCache just created is no more null, get the reference to the Context object associated to it (and corresponding to the primary key parameter) and, if not null and the response from the method isCascadeDeleteAfterSuperEJBRemove() is true (it probably means that the a cascade delete is performing, after a deletion of an EJb object has occurred), return a null object.

Finally, return an *EjbLocalObject*, getting its reference from the correspondent client.

1.7. Functional Role – Method 5

The task of this method is to remove an EJB (So at the end he will be in state: DESTROYED).

The method has the following parameters:

- **EJBLocalRemoteObject ejbo**: The reference to the object.
- **Method removeMethod**: The reference to the used method to remove the EJB.
- **boolean local**: A boolean value that express if the value is local.

Lines from 1101 to 1105 create an EjbInvocation with the given parameters.

Lines from 1110 to 1112 check the used remove method.

Lines from 1114 to 1123 is a general pattern "Called from EJBObject/EJBHome before invoking on EJB".
It prepares the instance of EjbInvocation (*preInvoke*), then removes the given EJB (*removeBean*) and finally release the element to other potential action (*postInvoke*)

Lines from 1118 to 1120 save in the log eventual errors and the caught exception is stored inside the instance of EjbInvocation.

Lines from 1125 to 1140 manage the eventual caught exception (if i.exception == null then no exceptions happened) and forward the suitable exception to the caller of this method.

2. Issues found applying the checklist

2.1. Class

Lines 203 and 205:

- These two declarations should be inverted: it is required to define first protected variables (line 205) and then private variables (line 203).

Line 216:

- Attribute name is divided from its respective type using one space and two TABs, instead of just one space.

Lines 222, 254, 255, 257:

- Attribute name is divided from its respective type using one TAB, instead of one space.

Line 257:

- A private class variable is defined at this line, but, in the previous lines, some protected variables have been defined: the required order is first public variables, then protected ones and finally the private ones.

2.2. Method 1

Lines from 697 to 698:

- There is an if statement so curly brackets are needed;

Line 768:

- The declaration of the variable status respects the standard because it is at the beginning of the else-if's block;

Lines from 772 to 774:

- There is an if statement so curly brackets are needed;

Lines from 775 to 776:

- There is an else statement so curly brackets are needed;

Line 788:

- In case of exception into the try-catch statement, the exception type is not written into the log;

EntityContextImpl javadocs do not cover all the methods that are implemented into the class.

2.3. Method 2

Lines from 818 to 820:

- The "if" statement is not surrounded by curly braces.

Lines 819-820:

- these two lines together does not exceed 120 characters, and it is more practical keeping them on the same line.

Line 827:

- the line length is more than 80 characters, but less than 120.
  A good alternative could be:
  containerStateManager.attachObject(inv,
  (EJBContextImpl)inv.context,ejbObjImpl, null);

Line 840:

- there is an object declared here (EntityContextImpl context). But, this is not the beginning of a block. This object should have been declared just after line 817 and initialized to null, and, at line 840 should be associated to *inv.context* in this way:
  context = (EntityContextImpl)inv.context;

2.4. Method 3

Lines: 937-948, 952-967, 970-977:

- It is an example of brutish programming. The code is repeated 3 times to do the same function. It is much better to create a private method doing the same thing. *(See the attached file: Method3.java)*

Lines from 940, 942, 959, 961, 971, 973:

- No curly brackets were used in the if-else.

Line 938:

- The variable *ref* is declared and not initialized. In future with a possible modify of the method can create a *NullPointerException*. It's better to initialize the variable to null (or to a default value).

Lines from 944 to 946:

- In the two branch of the if-else it's returned always the same variable with different value.
  It's much better to modify only the variable and return it outside that if-else. In this way the code is more extendible if another branch is added.

2.5. Method 4

Lines from 944 to 1033:

- the comment lines before the code of the method are indented in a bad way; for example, at the beginning of each line there are 5 spaces before "*" instead of 4. Moreover, in the text, 1 or 2 spaces are used to indent (instead of 3 or 4).

Lines from 1042 to 1060:

- these lines are part of a unique if statement. The indentation is not good at all. *(Attached we report the modified method in which it is possible to see also this statement indented in a better way. See Method4.java)*

Line 1050:

- this line should be indented: 4 more spaces are required at the beginning of this line, because this is the second part of the previous line.

Lines from 1051 to 1058:

- this is another if statement indented in an unintelligible way. *(See Method4.java the rewritten method attached.)*

Lines 1053:

- 3 TABs are used to indent instead of "4 spaces" for 3 times.

Line 1055:

- two TABs are used to indent at the beginning of this line, instead of "4 spaces" two times.

Line 1059:

- at the beginning of this line, a TAB is used to indent, instead of 4 spaces.

Lines 1059 and 1062:

- This line is executed anyway (unless the condition of the if statement at line 1052 is true. In this case the method will return null). Thus, it is useless to repeat the code. It is enough to keep line 1062, but it is absolutely better to erase line 1059.

2.6. Method 5

Line 1101:

- Declared a variable length 1 character *(i)*, a more significant name is preferable. (Ex. ejbInvocation)

Lines from 1101 to 1105:

- Create a method that returns the created object.
  Makes the method shorter and more readable, it's the base of the principle of "Divide et impera".

Line 1110:

- The name of the class should be written with the initial letter in uppercase.

Lines from 1111, 1112:

- Create a method that check the class and returns a boolean value instead of assign the result of a local comparison.
  It's clearer so it's easier to read and there can be less misunderstanding.
- The comparison should be made using equals and not the operator "==".

3. Other problems

Method 1:

In the comments there are not statement declaring parameters ("@param"), that should be used to generate Javadoc in the standard way.

Line 694: It is better to define a GET method to acquire the context's value from the invocation. It is possible to cast the context because the class EntityContextImpl implements the interface ComponentContext;

Line 729: It is better to have a GET method to acquire the startsWithCreate's value from invocationInfo contained in invocation;

Method 2:

In the comments there are not statement declaring parameters ("@param"), that should be used to generate Javadoc in the standard way.

Line 810: As written in the comment, this method is called from the generated "HelloEJBHomeImpl" create* method, but we were not able to find an object called in this way in the documentation.

Lines 824: "internalGetEJBObjectImpl" method was not found in the Javadocs, but it is actually in the class at line 1904.

Line 847: "setDirty" method not found in the Javadoc of the class EntityContextImpl.

Method 3:

In the comments there are not statement declaring parameters ("@param"), that should be used to generate Javadoc in the standard way.

Method 4:

Line 995: the comment at this line ends with five suspension points. It is useless and not good to read. A single suspension point is enough.

Line 1020: There is a grammatical mistake in the comment at this line: it is written "hcen" instead of "hence"

Lines 1012-1021: in the comment three examples of use of this method are listed: "example 2" is written two times. It should be written "example 3" at line 1021.

Moreover, a blank line should be after line 1020, in order to divide two different examples, as done after line 1010 to divide example 1 and example 2.

Line 1049: The object "ActiveTxCache" was not found in the documentation.

Line 1055: The method isCascadeDeleteAfterSuperEJBRemove() has not any comment associated in the Javadoc of entityContextImpl class.

Method 5:

In the comments there are not statement declaring parameters ("@param"), that should be used to generate Javadoc in the standard way.

Lines from 1102 to 1105: Variables are public so they are modified and used without setter and getter. It is better use private variables with public getter and setter.