POLITECNICO
MILANO 1863

Software Engineering 2 A.A. 2015-16

Inspection Document

for

MyTaxiService

by Francesco Picciotti (854021)

and Marco Travaglini (859186)

Table of Contents

Revision History

Date	Reason For Changes	Version
03/01/2016	First submission of the document	1.0

1. Classes

The methods assigned us belong all to the class BaseContainer.

- *Class: BaseContainer.java*
- *Package: appserver/ejb/ejbcontainer/src/main/java/com/sun/ejb/containers*
- *Methods:*
  - *postInvoke (EjbInvocation inv, boolean doTxProcessing)*
  - *addSchedule (TimerPrimaryKey timerId , EJBTimerSchedule ts)*
  - *processEjbTimeoutMethod (Method method)*
  - *mapRemoteException (EjbInvocation inv)*

2. Functional role

The inspected class BaseContainer is an abstract class that implements Container, EjbContainerFacade and JavaEEContainer.
Because of the class's size, it has more than 5000 lines of code, for understanding the functional role and meaning of the class, we have read the Javadoc not only of the inspected class, even of the main classes used inside BaseContainer and all the interfaces that are implemented by it. Moreover, we have not focused only on the behavior of the assigned methods but we have attempted to understand to a certain level of granularity the features of the whole class.
The class contains ejb instances and implements the logic for managing ejb invocations in every aspects; in fact, it is responsible for their lifecycle, state management, concurrency, transaction, naming and security.

❖ postInvoke

This method is called after an invocation in order to check the result and, based on it, it performs proper actions to ensure the correct working. It is done analyzing the state of the container, the invocation, the ejb and exception linked to the invocation, reporting in the logger more information about the exception, after throwing the right one or mapping it.

❖ addSchedule

It checks whether the method linked to the TimerSchedule, which is a parameter, is among the schedules currently saved and in this case, it adds the method to a hash map, which contains the links between the timer and the method, identifying it with a timer primary key.

- **processEjbTimeoutMethod**

This method is used when a timer related to a method expires and it tries to set the timeout accessible checking the presence of the security manager because in that case the action must be done in a privileged way, otherwise a security exception is thrown.

- **mapRemoteException**

It provides the correct mapping of the exception related to the ejb invocation that is passed as parameter, based on the kind of the invocation such like whether it is asynchronous, remote, local or if it belongs to the business interface. Even more, it reports the outcome and the mapped exception in the logger.

3. Issues

In the declaration of class's attributes it is not respected the following order: public, protected and private.
In the same way, the order of the methods does not agree with the correct one described above. Moreover, in the java source file, there are four classes, one of them is BaseContainer, which is public, and the other classes are final and public anyway but according the checklist a java source file should contain exactly one public class.
Furthermore in the whole class it is used the Kernighan and Ritchie bracing style but in the first if block inside the class's constructor is adopted the Allman style.
In addition, the Javadoc is not properly done in the whole java source file, wrapping the observed class, and it is not self-explaining. In some cases it is not complete, such like in addSchedule and processEjbTimeoutMethod, and in other cases it is not done at all, such like in postInvoke and mapRemoteException.

❖ **Lines 260-276 and 287**

```
private static final int EJB_INTF_METHODS_LENGTH = 16;
static final int EJBHome_remove_Handle      = 0;
static final int EJBHome_remove_Pkey        = 1;
static final int EJBHome_getEJBMetaData     = 2;
static final int EJBHome_getHomeHandle      = 3;
static final int EJBLocalHome_remove_Pkey   = 4;
static final int EJBObject_getEJBHome       = 5;
protected static final int EJBObject_getPrimaryKey    = 6; //TODO - move related to entity-container
static final int EJBObject_remove           = 7;
static final int EJBObject_getHandle        = 8;
static final int EJBObject_isIdentical      = 9;
static final int EJBLocalObject_getEJBLocalHome = 10;
protected static final int EJBLocalObject_getPrimaryKey   = 11; //TODO - move related to entity-container
static final int EJBLocalObject_remove      = 12;
static final int EJBLocalObject_isIdentical = 13;
static final int EJBHome_create             = 14;
static final int EJBLocalHome_create        = 15;
```

```
private static final byte[] homeInstanceKey = {HOME_KEY};
```

In the parts shown above and others not reported, the constants do not use uppercase words and in some cases there is not the "_", underscore letter, between words.

3.1 postInvoke

❖ **Lines 2049-2054:**

```
if (containerState != CONTAINER_STARTED) {
    throw new EJBException(localStrings.getLocalString(
            "ejb.container_not_started",
            "Attempt to invoke when container is in {0}",
            containerStateToString(containerState)));
}
```

In this first if-block there are three issues, the first is a presence of a tabulation before the if statement and the second is that there are more then four spaces (exactly eight) for indentation where the constructor's parameter is split (lines 2051-2053). The last one is about the EJBException, which is thrown but not handled anyway, thus no try-catch block or throws declaration in the method's signature.

❖ **Lines 2057-2062:**

```
inv.setDoTxProcessingInPostInvoke(doTxProcessing);
if (inv.mustInvokeAsynchronously()) {
    EjbAsyncInvocationManager asyncManager =
            ((EjbContainerUtilImpl) ejbContainerUtilImpl).getEjbAsyncInvocationManager();
    asyncManager.submit(inv);
    return;
}
```

Into the if-block there is another wrong indentation (about eight spaces) after the equal character in the assignment.

❖ **Lines 2064-2090:**

```
if ( inv.ejb != null ) {

    if (! inv.useFastPath) {
        invocationManager.postInvoke(inv);
        delistExtendedEntityManagers(inv.context);
    } else {
        doTxProcessing = doTxProcessing && (inv.exception != null);
    }

    try {
        if( doTxProcessing ) {
            postInvokeTx(inv);
        }
    } catch (Exception ex) {
        _logger.log(Level.FINE, "Exception occurred in postInvokeTx  : [{0}]", ex);
        if (ex instanceof EJBException)
            inv.exception = (EJBException) ex;
        else
            inv.exception = new EJBException(ex);
    }

    releaseContext(inv);
}
```

In this method's part there is an unhandled exception that might be thrown from the call of postInvoke(inv), it should be surrounded by a try-catch block or should be stated in the method declaration. In this case it might be:
void postInvoke(EjbInvocation inv,boolean doTxProcessing) throws InvocationException.
Moreover in the catch-block, braces are missing in the body of both if and else statements, indeed the overall try-catch block can be pushed into the if(doTxProcessing), because of the exception to be catched might be thrown by the postInvokeTx(inv) which is into the referred if statement.
Correspondingly, in the last line of the presented code, the method releaseContext throws an exception, which is not handled (no try-catch or throws declaration).

❖ Lines 2095-2104

```
// Log system exceptions by default and application exceptions only
// when log level is FINE or higher.

if( isSystemUncheckedException(inv.exception) ) {
    _logger.log(Level.WARNING, SYSTEM_EXCEPTION, new Object[]{ejbDescriptor.getName(), inv.beanMethod});
    _logger.log(Level.WARNING, "", inv.exception);
} else {
    _logger.log(Level.FINE, "An application exception occurred during an invocation on EJB {0}, method: {1}", new Object[]{ejbDescriptor.getName(), inv.beanMethod});
    _logger.log(Level.FINE, "", inv.exception);
}
```

In this part of code, there are comments that should explain, but are not understandable. Besides the first statement after the else is long about 161 characters, it should be split in one more line; similarly the first line after the if should be split too because it is long 100 characters.

❖ Lines 2106-2114 and 2131-2139

```
if( protocolMgr != null ) {
    // For remote business case, exception mapping is performed
    // in client wrapper.
    // TODO need extra logic to handle implementation-specific ejb exceptions
    // (ParallelAccessEXCeption etc. that used to be handled by iiop glue code
    inv.exception = mapRemoteException(inv);
}

        /*TODO
        if ( AppVerification.doInstrument()) {
            // need to pass the method, exception info,
            // and EJB descriptor to get app info
            AppVerification.getInstrumentLogger().doInstrumentForEjb(
            ejbDescriptor, inv.method, inv.exception);

        }
        */
```

In those code fragments there are commented TODO that should not be in final code, even more the second part of code is an entire if block commented out.

3.2 addSchedule

❖ **Lines 2207-2213**

```
if (m.getName().equals(ts.getTimerMethodName()) &&
        m.getParameterTypes().length == ts.getMethodParamCount()) {
    scheduleIds.put(timerId, m);
    if( _logger.isLoggable(Level.FINE) ) {
        _logger.log(Level.FINE, "Adding schedule: " +
                ts.getScheduleAsString() + " FOR method: " + m);
    }
}
```

Inside the inner if block, the second line in which the logger's statement is split has eight whitespace characters for indentation instead of at most four; in the outer if there is the same wrong indentation in the second line in which the if's condition is divided.

3.3 processEjbTimeoutMethod

❖ **Lines 2223-2225 and 2232**

```
if( (params.length == 0 ||
     (params.length == 1 && params[0] == javax.ejb.Timer.class)) &&
     (method.getReturnType() == Void.TYPE) ) {
```

```
if(System.getSecurityManager() == null) {
```

Both parts of code contains two if statement in which the condition has the "==" operator instead of the use of equals method. In particular, referring to the first part of code, this usage could be avoided for the initial comparisons between integers.
Moreover in the "if" clause it is used a class that should be imported in the beginning of the BaseContainer class.

❖ **Lines 2231-2245**

```
if(System.getSecurityManager() == null) {
    if( !ejbTimeoutAccessible.isAccessible() ) {
        ejbTimeoutAccessible.setAccessible(true);
    }
} else {
    AccessController.doPrivileged(
            new java.security.PrivilegedExceptionAction() {
                public java.lang.Object run() throws Exception {
                    if( !ejbTimeoutAccessible.isAccessible() ) {
                        ejbTimeoutAccessible.setAccessible(true);
                    }
                return null;
            }
    });
}
```

The part of code shown here, in the else block where is created a new object through the keyword "new" too many whitespace characters are used, they are exactly eight instead of four.
Still in the same block, there is the usage of classes not imported in the BaseContainer's beginning where there are all the used imports.
In addition in the anonymous class, when it is re-implemented the run method the keyword "@Override" is missing and especially the run method's returned value is not used properly. Thus, it returns null and it is a bad practice in the java programming but even more, it may cause the creation of a nullpointer exception that is not catched or declared with a "throws" keyword.
Furthermore, there is duplicate of code, the "if" that checks whether the timeout is accessible and sets it to true, it can be wrapped into a method and the whole part of code above should be replaced with a try-catch block. The try clause should contain the call of the method wrapping the duplicated code and the catch clause should handle the exception thrown by the setAccessible method, in case there is a SecurityManager, calling the method that should contain the duplicated code in a privileged way, in order to bypass the SecurityManager.

❖ **Lines 2247-2252**

```
  else {
    throw new EJBException(localStrings.getLocalString(
        "ejb.invalid_timeout_method",
        "Invalid @Timeout or @Schedule signature for: {0} @Timeout or @Schedule method must return void and be a no-arg method or take a single javax.ejb.Timer param",
        method));
}
```

Obviously the third line inside the else block is too long, in fact it is long about 159 characters and it should easily split in at least two lines

3.4 mapRemoteException

❖ **Lines 2283-2290**

```
if( mappedException == originalException) {

    if( originalException instanceof EJBException ) {
        mappedException = new RemoteException
            (originalException.getMessage(),originalException);
    }

}
```

In this portion of code the two nested ifs can be merged in a unique one having the two conditions in "&&" with each other; this can be done because the inner if doesn't have any else branch and also there's no statement between the nested blocks.
Furthermore, inside the first condition instead of the usage of "==" is advisable adopt the method equals.

❖ **Lines 2305-2323**

```
if( mappedException == originalException) {

    if( inv.isBusinessInterface ) {

        if(originalException instanceof EJBException) {
            mappedException = new InternalEJBContainerException
                (originalException.getMessage(), originalException);
        }

    } else {
        if( originalException instanceof EJBException ) {
            mappedException = new RemoteException
                (originalException.getMessage(), originalException);
        }
    }
}
```

In the first if from the top, it is used the "==" operator instead of the method equals.
Besides the "ifs", which are inside the second if-else that is at first level of nesting, can be swapped their nesting to the outer one. In fact there are two if with identical condition inside both the if and else branches, so it had better to first check whether the original exception is instance of EJBException and then create the proper exception if the invocation is business interface or not.

❖ **Lines 2329-2331**

```
_logger.log(Level.FINE, "Mapped original remote exception " +
        originalException + " to exception " + mappedException +
        " for " + inv);
```

The reported statement is rightly split in more than one line but the number of whitespace characters for the indentation is eight instead of, at most, four.

4. Other problems

❖ *The class variables displayed below and many later on are initialized, when are declared, to null. Besides being a useless operation, in fact in Java when a not primitive variable is declared his reference is automatically null, it is not a proper initialization such like instantiating an object or setting his value to a reference of an existing one.*

```
protected ClassLoader loader = null;
protected Class ejbClass = null;
protected Class sfsbSerializedClass = null;
protected Method ejbPassivateMethod = null;
protected Method ejbActivateMethod = null;
protected Method ejbRemoveMethod = null;
private Method ejbTimeoutMethod = null;
```

❖ *The class variable methodMonitorMap is neither initialized nor used in the whole class. Moreover in the following attributes' declaration are inserted many useless whitespaces in order to give a strange and normally unused way of indentation.*

```
protected HashMap                methodMonitorMap;
protected boolean                monitorOn = false;

protected EjbMonitoringStatsProvider    ejbProbeListener;
protected EjbMonitoringProbeProvider    ejbProbeNotifier;
protected EjbTimedObjectStatsProvider   timerProbeListener;
protected EjbTimedObjectProbeProvider   timerProbeNotifier;
protected EjbPoolStatsProvider          poolProbeListener;
protected EjbCacheProbeProvider         cacheProbeNotifier;
protected EjbCacheStatsProvider         cacheProbeListener;

protected ContainerInfo                 containerInfo;
```

- *The method delistExtendedEntityManagers is empty and it is not stated in the JavaDoc, it may have sense but it had better to be defined as abstract method otherwise there could be the risk of leaving blank or not re-implemented in the subclasses causing misleading and wrong behaviors.*

```
protected void delistExtendedEntityManagers(ComponentContext ctx) {
    // Do nothing in general case
}
```

- *In this part of code, there is a cast of the attribute ejbContainerUtilImpl but before this it should be checked that the dynamic type of the attribute is the same of the cast. Otherwise, it could create an exception since the static type of the attribute have not the method called.*

```
inv.setDoTxProcessingInPostInvoke(doTxProcessing);
if (inv.mustInvokeAsynchronously()) {
    EjbAsyncInvocationManager asyncManager =
            ((EjbContainerUtilImpl) ejbContainerUtilImpl).getEjbAsyncInvocationManager();
    asyncManager.submit(inv);
    return;
}
```

5. Appendix

5.1 Tools

- *Microsoft Word: used for creating this document.*
- *Notepad++ & Android Studio: used for inspecting the assigned code.*
- *Microsoft OneDrive: use for sharing files among the group.*

5.2 Hours of work

- *Francesco: 20 hours.*
- *Marco: 20 hours.*