



# POLITECNICO MILANO 1863

Software Engineering 2 Project

## MyTaxiService

PART 3:  
Code Inspection

Valeria Deciano 858479  
Fabio Calabretta 852717

## **SUMMARY**

- 1. Classes that were assigned to the group**
- 2. Functional role of assigned set of classes**
- 3. List of issues found by applying the checklist**
- 4. Any other problem you have highlighted**

## 1. Methods that were assigned to the group

**Name:**forceDestroyBean( EJBContextImpl ctx )

**Start Line:**1267

**Location:** appserver/ejb/ejbcontainer/src/main/java/com/sun/ejb/containers/StatefulSessionContainer.java

```
protected void forceDestroyBean(EJBContextImpl ctx) {
    SessionContextImpl sc = (SessionContextImpl) ctx;

    synchronized (sc) {
        if (sc.getState() == EJBContextImpl.BeanState.DESTROYED)
            return;

        // mark context as destroyed so no more invocations happen on it
        sc.setState(BeanState.DESTROYED);

        cleanupInstance(ctx);

        if (_logger.isLoggable	TRACE_LEVEL)) {
            _logger.log	TRACE_LEVEL, "[SFSBContainer] (Force)Destroying "
                + "session: " + sc.getInstanceKey());
        }

        Transaction prevTx = sc.getTransaction();
        try {
            if (prevTx != null && prevTx.getStatus() !=
                Status.STATUS_NO_TRANSACTION) {
                prevTx.setRollbackOnly();
            }
        } catch (SystemException ex) {
            throw new EJBException(ex);
        } catch (IllegalStateException ex) {
            throw new EJBException(ex);
        }

        // remove the bean from the session store
        Object sessionKey = sc.getInstanceKey();
        sessionBeanCache.remove(sessionKey, sc.existsInStore());

        if (isRemote) {

            if (hasRemoteHomeView) {
                // disconnect the EJBObject from the context and vice versa
                EJBObjectImpl ejbObjImpl = sc.getEJBObjectImpl();
                ejbObjImpl.clearContext();
                ejbObjImpl.setRemoved(true);
                sc.setEJBObjectImpl(null);
            }
        }
    }
}
```

```

        // disconnect the EJBObject from the ProtocolManager
        // so that no remote invocations can reach the EJBObject
        remoteHomeRefFactory.destroyReference
            (ejbObjImpl.getStub(), ejbObjImpl.getEJBObject());
    }

    if (hasRemoteBusinessView) {

        EJBObjectImpl ejbBusinessObjImpl =
            sc.getEJBRemoteBusinessObjectImpl();
        ejbBusinessObjImpl.clearContext();
        ejbBusinessObjImpl.setRemoved(true);
        sc.setEJBRemoteBusinessObjectImpl(null);

        for (RemoteBusinessIntfInfo next :
            remoteBusinessIntfInfo.values()) {
            // disconnect from the ProtocolManager
            // so that no remote invocations can get through
            next.referenceFactory.destroyReference
                (ejbBusinessObjImpl.getStub
                    (next.generatedRemoteIntf.getName()),
                    ejbBusinessObjImpl.getEJBObject
                        (next.generatedRemoteIntf.getName()));
        }
    }
}

if (isLocal) {
    if (hasLocalHomeView) {
        // disconnect the EJBLocalObject from the context
        // and vice versa
        EJBLocalObjectImpl localObjImpl =
            (EJBLocalObjectImpl) sc.getEJBLocalObjectImpl();
        localObjImpl.clearContext();
        localObjImpl.setRemoved(true);
        sc.setEJBLocalObjectImpl(null);
    }
    if (hasLocalBusinessView) {
        // disconnect the EJBLocalObject from the context
        // and vice versa
        EJBLocalObjectImpl localBusinessObjImpl =
            (EJBLocalObjectImpl) sc.getEJBLocalBusinessObjectImpl();
        localBusinessObjImpl.clearContext();
        localBusinessObjImpl.setRemoved(true);
        sc.setEJBLocalBusinessObjectImpl(null);
    }
    if (hasOptionalLocalBusinessView) {
        EJBLocalObjectImpl optionalLocalBusinessObjImpl =
            (EJBLocalObjectImpl) sc.getOptionalEJBLocalBusinessObjectImpl();
        optionalLocalBusinessObjImpl.clearContext();
        optionalLocalBusinessObjImpl.setRemoved(true);
        sc.setOptionalEJBLocalBusinessObjectImpl(null);
    }
}
}

```

```

destroyExtendedEMsForContext(sc);

// tell the TM to release resources held by the bean
transactionManager.componentDestroyed(sc);

    }
}

```

**Name:**releaseContext( EjbInvocation inv )

**Start Line:**1878

**Location:** appserver/ejb/ejbcontainer/src/main/java/com/sun/ejb/containers/StatefulSessionContainer.java

```

public void releaseContext(EjbInvocation inv) {
    SessionContextImpl sc = (SessionContextImpl) inv.context;

    // Make sure everything is within try block so we can be assured that
    // any instance lock is released in the finally block.
    try {
        // check if the bean was destroyed
        if (sc.getState() == BeanState.DESTROYED)
            return;

        // we're sure that no concurrent thread can be using this
        // context, so no need to synchronize.
        Transaction tx = sc.getTransaction();

        // If this was an invocation of a remove-method
        if (inv.invocationInfo.removalInfo != null) {

            InvocationInfo invInfo = inv.invocationInfo;
            EjbRemovalInfo removeInfo = invInfo.removalInfo;

            if (retainAfterRemoveMethod(inv, removeInfo)) {
                _logger.log(Level.FINE, "Skipping destruction of SFSB "
                    + invInfo.ejbName + " after @Remove method "
                    + invInfo.method + " due to (retainIfException"
                    + " == true) and exception " + inv.exception);
            } else {
                try {
                    destroyBean(inv, sc);
                } catch (Throwable t) {
                    _logger.log(Level.FINE, "@Remove.preDestroy exception",
                        t);
                }

                // Explicitly null out transaction association in bean's context.
                // Otherwise, forceDestroyBean() will mark that tx for rollback,
                // which could incorrectly rollback a client-propagated transaction.
                sc.setTransaction(null);
            }
        }
    }
}

```

```

        forceDestroyBean(sc);

        // The bean has been destroyed so just skip any remaining processing.
        return;
    }
}

if (tx == null || tx.getStatus() == Status.STATUS_NO_TRANSACTION) {
    // The Bean executed with no tx, or with a tx and
    // container.afterCompletion() was already called.
    if (sc.getState() != BeanState.READY) {
        if (sc.isAfterCompletionDelayed()) {
            // ejb.afterCompletion was not called yet
            // because of container.afterCompletion may have
            // been called concurrently with this invocation.
            if (_logger.isLoggable	TRACE_LEVEL)) {
                logTraceInfo(inv, sc,
                    "Calling delayed afterCompletion");
            }
            callEjbAfterCompletion(sc, sc.getCompletedTxStatus());
        }

        if (sc.getState() != BeanState.DESTROYED) {
            // callEjbAfterCompletion could make state as DESTROYED
            sc.setState(BeanState.READY);
            handleEndOfMethodCheckpoint(sc, inv);
        }
    }

    if ((sc.getState() != BeanState.DESTROYED)
        && isHAEnabled) {
        syncClientVersion(inv, sc);
    }
} else {
    if ((sc.getState() != BeanState.DESTROYED)
        && isHAEnabled) {
        syncClientVersion(inv, sc);
    }
    sc.setState(BeanState.INCOMPLETE_TX);
    if (_logger.isLoggable	TRACE_LEVEL)) {
        logTraceInfo(inv, sc, "Marking state == INCOMPLETE_TX");
    }
}

} catch (SystemException ex) {
    throw new EJBException(ex);
} finally {
    releaseSFSBSerializedLock(inv, sc);
}
}

```

## 2. Functional role of assigned class and set of methods

### class StatefulSessionContainer

This public class is the dual of the StatelessSessionContainer and provides container functionality specific to stateful SessionBeans.

At deployment time, one instance of the StatefulSessionContainer is created for each stateful SessionBean type in a JAR.

The 5 states of a Stateful EJB (Enterprise JavaBean)

- PASSIVE : has been passivated
- READY : ready for invocations, no transaction in progress
- INVOKING : processing an invocation
- INCOMPLETE\_TX : ready for invocations, transaction in progress
- DESTROYED : does not exist

Note that an EJB can be in only 1 state at a time.

### forceDestroyBean( EJBContextImpl ctx )

The main role of this method is to discard the SFSB (StateFul SessionBean) and definitely destroy it. Also, it makes a rollback for each transaction that was associated with called from 'removeBean', 'timeoutBean' and 'BaseContainer.postInvokeTx'.

Then it checks which kind of view had the bean, that can be Remote, Local or both. For each of them there is a second distinction between Home and Business, in such a way that we have all this type of view:

- RemoteHomeView
- RemoteBusinessView
- LocalHomeView
- LocalBusinessView

For each of them the method tries to disconnect the EJBLocalObject from the context and vice versa. Finally it tells to the Transaction Manager to release the resources held by the bean.

### releaseContext( EjbInvocation inv )

This method mainly ensures to release the lock (formerly acquired with a "\_getContext" call) holding by the SFSB for a particular invocation. Before do this, the method checks if the invocation passed through parameter was a remove-method invocation. If it is, then it verifies whether destroy the bean (and consequently ends the process) or retain it. Finally if the bean is not *destroyed* and the "*high availability*" (as known as HA) of the session is enabled, the method tries to synchronize the client version.

### 3. List of issues found by applying the checklist

Point 33 and 11 of the checklist states:

*33. Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces “{” and “}”). The exception is a variable can be declared in a ‘for’ loop.*

*11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces. Example:*

*Avoid this:*

*if ( condition )*

*doThis();*

*Instead do this:*

*if ( condition )*

*{*

*doThis();*

*}*

#### **1. forceDestroyBean**

We detect the declarations with wrong position and the missing braces.

```
protected void forceDestroyBean(EJBContextImpl ctx) {
    SessionContextImpl sc = (SessionContextImpl) ctx;

    synchronized (sc) {
        if (sc.getState() == EJBContextImpl.BeanState.DESTROYED)
            return;

        sc.setState(BeanState.DESTROYED);

        cleanupInstance(ctx);

        if (_logger.isLoggable	TRACE_LEVEL)) {
            _logger.log	TRACE_LEVEL, "[SFSBContainer]
            (Force)Destroying " + "session: " + sc.getInstanceKey());
        }
    }
}
```



```

Transaction prevTx = sc.getTransaction();
try {
    if (prevTx != null && prevTx.getStatus() !=
        Status.STATUS_NO_TRANSACTION) {
        prevTx.setRollbackOnly();
    }
} catch (SystemException ex) {
    throw new EJBException(ex);
} catch (IllegalStateException ex) {
    throw new EJBException(ex);
}

// remove the bean from the session store
Object sessionKey = sc.getInstanceKey();
sessionBeanCache.remove(sessionKey, sc.existsInStore());
...
}

```

And we modify the code like the following.

```

protected void forceDestroyBean(EJBContextImpl ctx) {

    SessionContextImpl sc = (SessionContextImpl) ctx;

    synchronized (sc) {

        Transaction prevTx;
        Object sessionKey; (1)
        if (sc.getState() == EJBContextImpl.BeanState.DESTROYED){
            return;
        }

        ...

    }

    ...

}

```

## 2. releaseContext

For the same previous two points of the checklist the following piece of code...

```

public void releaseContext(EjbInvocation inv) {
    SessionContextImpl sc = (SessionContextImpl) inv.context;


```

```

    try {
        if (sc.getState() == BeanState.DESTROYED)
            return;

        Transaction tx = sc.getTransaction();
        ...
    }
    ...
}

```

 Note that it is useless to initialize explicitly the variables to "null" because Java does already automatically.  
..should be like this.

```

public void releaseContext(EjbInvocation inv) {
    SessionContextImpl sc = (SessionContextImpl) inv.context;

    try {
        Transaction tx;
        if (sc.getState() == BeanState.DESTROYED){
            return;
        }

        tx = sc.getTransaction();
        ...
    }
    ...
}

```

*13. Where practical, line length does not exceed 80 characters.*

From line 1912.

```

// Explicitly null out transaction association in bean's context.
// Otherwise, forceDestroyBean() will mark that tx for rollback,
// which could incorrectly rollback a client-propagated transaction.
sc.setTransaction(null);

forceDestroyBean(sc);

// The bean has been destroyed so just skip any remaining processing.
return;

```

80

Modified code:

```
// Explicitly null out transaction association in bean's
// context. Otherwise, forceDestroyBean() will mark that tx
// for rollback, which could incorrectly rollback a
// client-propagated transaction.
sc.setTransaction(null);

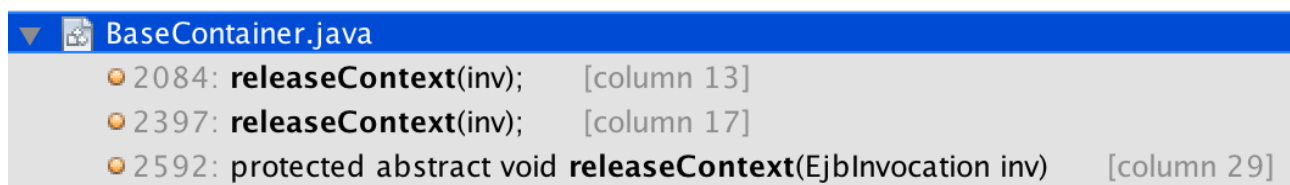
forceDestroyBean(sc);

// The bean has been destroyed so just skip any remaining
// processing.
return;
```

#### 4. Any other problem you have highlighted

```
/**
 * Called from preInvoke which is called from the EJBObject for
 * local and remote invocations.
 */
public void releaseContext(EjbInvocation inv) {
    ...
}
```

But *releaseContext* is used in BaseContainer.**postInvoke**(inv, doTxProcessing) [line 2084] and in BaseContainer.**authorize**(inv) [line 2397]



```
▼ BaseContainer.java
  2084: releaseContext(inv);    [column 13]
  2397: releaseContext(inv);    [column 17]
  2592: protected abstract void releaseContext(EjbInvocation inv)    [column 29]
```

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

```

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

if ( inv.ejb != null ) {
    // counterpart of invocationManager.preInvoke
    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);
}

if ( inv.exception != null ) {

    // Unwrap the PreInvokeException if necessary
    if ( inv.exception instanceof PreInvokeException ) {
        inv.exception =
            ((PreInvokeException)inv.exception).exception;
    }

    // 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);
    }

    if ( inv.isRemote ) {

        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);
        }

        // The most useful portion of the system exception is
        // logged above. Only log mapped form when log level is
        // FINE or higher.
        _logger.log(Level.FINE, "", inv.exception);

    } else {

        if( inv.isBusinessInterface ) {
            inv.exception =
                mapLocal3xException(inv.exception);
        }
    }

    /*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);
    }
    */

    if( _logger.isLoggable(Level.FINE) ) {
        _logger.log
            (Level.FINE, "Leaving BaseContainer::postInvoke:" + inv);
    }

```

```

    }
}

/**
 * Common code to handle EJB security manager authorization call.
 */
public boolean authorize(EjbInvocation inv) {

    // There are a few paths (e.g. authorizeLocalMethod,
    // authorizeRemoteMethod, Ejb endpoint pre-handler )
    // for which invocationInfo is not set. We get better
    // performance with the security manager on subsequent
    // invocations of the same method if invocationInfo is
    // set on the invocation. However, the authorization
    // does not depend on it being set. So, try to set
    // invocationInfo but in this case don't treat it as
    // an error if it's not available.
    if( inv.invocationInfo == null ) {

        inv.invocationInfo = getInvocationInfo(inv);

    }

    // Internal methods for 3.0 bean creation so there won't
    // be corresponding permissions in the security policy file.
    if( (inv.method.getDeclaringClass() == localBusinessHomeIntf)
        ||
        (inv.method.getDeclaringClass() == remoteBusinessHomeIntf) )
{
    return true;
}

    boolean authorized = securityManager.authorize(inv);

    if( !authorized ) {

        if( inv.context != null ) {
            // This means that an enterprise bean context was
            // created during the authorization call because of a
            // callback from a JACC enterprise bean handler. Since
            // the invocation will not proceed due to the
            // authorization failure, we need to release the
            // enterprise bean context.
            releaseContext(inv);
        }

        return authorized;
    }
}

```

The heading of the method could be like the following:

```
/**
 * Called from postInvoke if no exception is thrown and from
 * authorize if the enterprise bean context is not null and the
 * authorization is not successful.
 */
public void releaseContext(EjbInvocation inv) {
    ...
}
```