SE2 Inspection Document

Edoardo Giacomello Mattia Fontana

January 4, 2016

Contents

1 Assigned Classes and Methods

Assigned Class: BaseContainer.java **Location**: appserver/ejb/ejb-container/src/main/java/com/sun/ejb/containers/BaseContainer.java **Package**: com.sun.ejb.containers Methods to Inspect:

1. **Name**:mapLocal3xException(Throwable t)
   - **Start Line**:2337
2. **Name**: authorize(EjbInvocation inv)
   - **Start Line**:2362
3. **Name**:initializeEjbInterfaceMethods( )
   - **Start Line**:2408
4. **Name**:getJaccEjb(EjbInvocation inv)
   - **Start Line**:2676
5. **Name**:assertValidLocalObject(Object o)
   - **Start Line**:2725

2 Functional Roles

This section will explain what is the functional role of the class and methods we analysed and will describe the process that have been used in order to discover these functional roles.

2.1 WorkFlow

For getting a better understanding of the analysed component functional roles and the general context, the following steps have been followed:

1. Javadoc inspection of the assigned class, with respect to implemented interfaces, subclasses and implementers.
2. Reading of the document "Enterprise JavaBeansTM Specification Version 2.0", in particular of the section regarding the container contract and functionalities overview
3. Finding the usage of the methods to analyse by using the grep tool

4. Documentation inspection and usage analysis of caller methods and their classes

5. Documentation inspection of the methods that have been assigned for code review

6. Functional Inspection for the code of the methods that have been assigned

7. Definition of the Scope for the methods that have been assigned

8. Code inspection

2.2 Package overview

The package com.sun.ejb.containers provides all the classes needed for implementing an EJB container, which can be either **Stateful** or **Stateless**, an Entity Bean container, or Message Bean Container.
It also provides classes that implement the container Home interface, which defines the methods for the client to create, remove, and find EJB objects of the same type (EJBHomeImpl class).

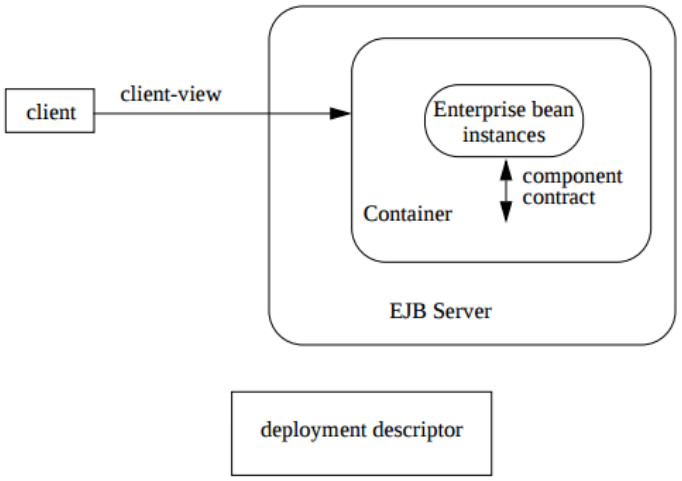
2.3 BaseContainer Class

In this section will be described the main scope of the class that contains the analysed methods.

The **BaseContainer** class implements the Container interfaces as stated in the ***EJB 2.0 specifications***. It hosts the code that is shared between the *Session Beans*, *Entity Beans* and *Message Driven Beans*.

The scope of this class is therefore to provide a common interface between the different types of Java Bean Containers. The context of operation can be inferred by the following diagram, included in the Java Bean Specification document:

Figure 1 Enterprise JavaBeans Contracts



Note that while the figure illustrates only a remote client running outside of the Container, the client-view APIs are also applicable to local clients and to remote clients that are enterprise Beans deployed in the same Container.

An analysis of the code revealed that this class in particular manages the object that contains an EJB Method invocation in several context such as Authorization, Initialization, Pre-Invoking, Post-Invoking etc.

2.3.1 Interfaces

This class implements directly the following interfaces:

Container : This interface is the main contract for a EJB Container implementation. In this case the container is a specific implementation of this interface (see BaseContainer subclasses) and it is responsible for managing the lifecycle, state management, concurrency, transactions etc, by interposing actions before and after invocations on EJBs. The methods that have been analysed are specified in this interface.

JavaEEContainer : The javadoc does not specifies a description for this interface, but the method names suggest that it provides some utility methods for all the JEE containers, such the retrieval of the component Id and the container descriptor.

EjbContainerFacade : This interface provides ejb-specific methods for iiop middleware integration, which is a protocol for distributed systems that supports the mapping between TCP/IP and Inter-Object Request Broker messages.

2.3.2 Subclasses

The **BaseContainer** class is derived by the following classes, each of them implementing a different type of EJB container.

EntityContainer : This class represents a container for an Entity Bean and It is responsible for their instances and lifecycle management. In particular, this type of container (*EJB Spec 2.0, section 10.5.9*) does not ensure that the instance has exclusive access to the state of the object in persistence storage, and the container must therefore synchronize the instance's state at the beginning of a transaction.

MessageBeanContainer This class provides container functionality specific to message-driven EJBs. At deployment time, one instance of the MessageDrivenBeanContainer is created for each message-driven bean in an application. (*Class Javadoc*)

StatefulSessionContainer This class provides container functionality specific to stateful SessionBeans. At deployment time, one instance of the StatefulSessionContainer is created for each stateful SessionBean type (i.e. deployment descriptor) in a JAR. (*Class Javadoc*)

StatelessSessionContainer This class provides container functionality specific to stateless SessionBeans. At deployment time, one instance of the StatelessSessionContainer is created for each stateless SessionBean type (i.e. deployment descriptor) in a JAR.
This container services invocations using a pool of EJB instances. An instance is returned to the pool immediately after the invocation completes, so the number of instances needed = number of concurrent invocations.

A Stateless Bean can hold open DB connections across invocations. Its assumed that the Resource Manager can handle multiple incomplete transactions on the same connection.

AbstractSingletonContainer Called from the JarManager at deployment time.

2.3.3 Class Body

The **BaseContainer** class includes the following nested Classes:

ContainerInfo This class contains strings for monitoring the container information.

ContainerType This enum specifies the type of the container, that can be Entity, MessageDriven, ReadOnly, Singleton, Stateful or Stateless

PreInvokeException This is a wrapper for the exceptions thrown from BaseContainer.preInvoke, so it indicates that the bean's method will not be called. *(from Javadoc)* The preInvokeMethod is a method which is called from the EJB home or object before the invocation of the bean method.

2.4 Terminology and other Components

This section contains all the specific terminology and components that have been referred to during the functional description of the analysed code.

2.4.1 Local and Remote Clients

From Oracle Documentation:
A **local client** has these characteristics.

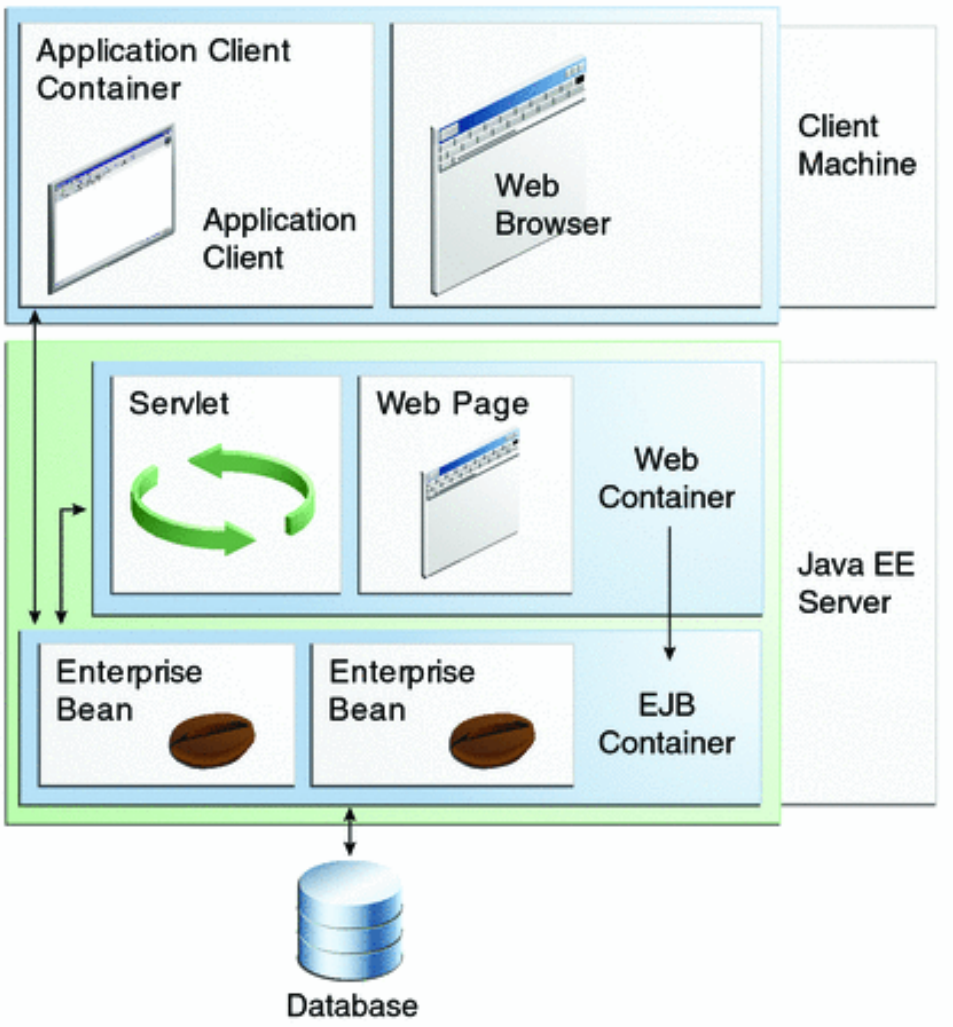
- It must run in the same application as the enterprise bean it accesses.
- It can be a web component or another enterprise bean.
- To the local client, the location of the enterprise bean it accesses is not transparent.

A **remote client** of an enterprise bean has the following traits.

- It can run on a different machine and a different JVM from the enterprise bean it accesses. (It is not required to run on a different JVM.)
- It can be a web component, an application client, or another enterprise bean.
- To a remote client, the location of the enterprise bean is transparent.
- The enterprise bean must implement a business interface. That is, remote clients may not access an enterprise bean through a no-interface view.

2.4.2 EJB Container

Containers are the interface between a component and the low-level platform-specific functionality that supports the component. Before it can be executed, a web, enterprise bean, or application client component must be assembled into a Java EE module and deployed into its container.
A more explicative description is given in the picture below:



2.4.3 EJB Home

From EJBHome javadoc:
The EJB Home is an interface that defines the methods that allow a remote client to create, find, and remove EJB objects.
The remote home interface is defined by the enterprise bean provider and implemented by the enterprise bean container.
Enterprise beans written to the EJB 3.0 and later APIs do not require a home interface.

2.4.4 EJB Local Object

From Oracle JavaDoc:
An enterprise bean's local interface provides the local client view of an EJB object. An enterprise bean's local interface defines the business methods

callable by local clients. The enterprise bean's local interface is defined by the enterprise bean provider and implemented by the enterprise bean container. Enterprise beans written to the EJB 3.0 and later APIs do not require a local interface that extends the EJBLocalObject interface. A local business interface can be used instead.

2.4.5 EJB Invocation

The EjbInvocation object contains the state associated with an invocation on an EJB or EJBHome (local/remote). It is usually created by generated code in *ObjectImpl and *HomeImpl classes. It is passed as a parameter to Container.preInvoke() and postInvoke(), which are called by the EJB(Local)Object/EJB(Local)Home before and after an invocation.

2.4.6 JACC: Java Authorization Contract for Containers

From Oracle Documentation: The Java Authorization Contract for Containers (JACC) specification defines a contract between a Java EE application server and an authorization policy provider. All Java EE containers support this contract.

The JACC specification defines java.security.Permission classes that satisfy the Java EE authorization model. The specification defines the binding of container access decisions to operations on instances of these permission classes. It defines the semantics of policy providers that use the new permission classes to address the authorization requirements of the Java EE platform, including the definition and use of roles.

2.5 Methods

All considered methods are implemented in the **BaseContainer** class.

2.5.1 mapLocal3xException

Visibility : Private Method.

Definition Class : **BaseContainer** class

Called from :

- method **postInvoke** in class **BaseContainer**, row **2124**
- method **mapRemoteException** in class **BaseContainer**, row **2292**

Usage Analysis :

- The first usage occurrence is into the **postInvoke** method of this same class. The postInvoke method is a method which is called from the EJB Home or Container after the invocation of the bean method. In the case an exception is raised and the invocation is not remote, the exception dynamic type is mapped by the **mapLocal3xException** method and stored into the **EJBInvocation** object.
- The second usage occurrence is into the **mapRemoteException** method of this same class. The mapRemoteException method checks if a remote exception invocation is asynchronous: in that case, as stated in comment lines, we are sure that the exception is raised by a remote business interface and not from a 2.x client, so it has to be mapped as a local exception by the **mapLocal3xException** method.

Functional Description : This method consists in a check over the dynamic type of a **Throwable** taken as input. If the instance matches one of the exception type defined, it is re-instantiated as a corresponding non-local exception and returned.

Scope : Although no javadoc is available for this method, through the usage and code analysis, with respect to the Oracle Javadoc of the package javax.ejb it has been possible to deduce that this is an helper method, which maps a Local EJB exception of the 3.x version into another corresponding exception that could be sent to the client.

2.5.2 authorize

Visibility : Public Method

Definition Class : **Container** interface

Called from :

- method **preInvoke** in class **BaseContainer**, row **1959**
- method **authorizeLocalMethod** in class **BaseContainer**, row **2162**
- method **authorizeRemoteMethod** in class **BaseContainer**, row **2185**
- method **invoke** in class **WebServiceInvocationHandler**, row **2185**
- method **authorizeWebService** in class **EjbInvocation**, row **675**

Usage Analysis :

- **preInvoke** is the method which is called from the EJB container before the invocation of the actual EJB method. It checks if the state of the method invocation is legitimate or it would lead to exceptions.
  The call of the authorize method is done in the context of security checking: if the authorize method returns false, an exception is raised which states that the client is not authorized to make that method invocation.
- the **authorizeLocalMethod** and **authorizeRemoteMethod** are called from the local/remote container home or object respectively in order to authorize the execution of a EJB method; the usage of the authorize method is similar to that in the previous point.
- **WebServiceInvocationHandler** is a proxy invocation handler for web service ejb invocations.
  It calls the authorize method for checking if the client is authorized to call a certain method through that proxy
- The **EjbInvocation** is the object that contains the state of an EJB Method invocation. No javadoc is available, but the name and the code suggest that the authorize method is used to authorize a web service method call by accessing the container that own the invocation itself.

Functional Description : The method first try to fetch the method invocation associated information and attaches it to the invocation object, because it would improve performance.
Then it checks if the called method has been called from the business home interface, in that case it will return true; if not the method will call the authorize method of the security manager. If the security manager doesn't authorize the invocation, its context is released.

Scope : The JavaDoc states that this method contains the common code for managing the security manager authorization call.
In practice, this method is useful to assert if the client who calls an EJB method is authorized to make that invocation.
By code inspection it is possible to understand that this method will make a check on the source of the called EJB method and authorize it automatically or invoke the security manager instead.

2.5.3 initializeEjbInterfaceMethods

Visibility : Private Method

Definition Class : **BaseContainer** class

Called from :

- **Constructor** of **BaseContainer** class, row **840**.

Usage Analysis :

- This method is called by the class Constructor during the initialization process of a Container.

Functional Description : This method creates an array of Methods that will contains all methods of the EJB interface, according to its type (Local or Remote, Stateless or Stateful).

Scope : This method adds by reflection the interface methods that the BaseContainer class has implemented and assign the produced array of methods to the local ejb home or local object.

2.5.4 getJaccEjb

Visibility : Public method

Definition Class : **Container** interface.

Called from :

- method **getJaccEjb** in class **EjbInvocation**, row **368**

- Indirectly by method **getEnterpriseBean** in class **EJBPolicyContextDelegate**, row **60** (see Usage Analysis)

Usage Analysis :

- The BaseContainer getJaccEjb method is only called in the method with the same name which below to the invocation itself. The javadoc of the invocation version of the method states that the user shall call the getJaccEjb method on the invocation object rather than directly on the EJB field, but it just call the Container method and return its value.

- The EJBPolicyContextDelegate is a delegate for the Policy Context and it calls getJaccEjb for returning the bean that is owned by the invocation object.

Functional Description : The operation flow of this method is based on several assumptions that are specified in the comments.
First of all the method make a check on the invocation passed as parameter, it has to be a business method invocation done through a remote, local or serviceEndpoint interface. Then if the context for the invocation has not been set, it is done and the ejb for that context is returned. There an important consideration about the accessed variable is made in the comments, but it doesn't affect the way the method works.

Scope : This method retrieves the Java Bean from the invocation context. This is necessary for the JACC policy provider (see Terminology section).

2.5.5 assertValidLocalObject

Visibility : Public method

Definition Class : **Container** interface

Called from :

- method **assertValidLocalObject** in class **SunContainerHelper**, row **240**
- Indirectly, method **assertValidLocalObject** in class **CMPHelper**, row **234**
- Indirectly, method **assertValidLocalObjectImpl** in class **JDOEJB20HelperImpl**, row **243**

Usage Analysis :

- This method is used in helper class that manages the Container-Managed persistance for java beans.
  In particular, the JDOEJB20HelperImpl is an helper class that is useful to convert persistance-capable beans from and to single object and collection of these.

Functional Description : This method receives as input an object, and it will raise an exception in the case the object is not a local valid one, or just return otherwise. It starts checking if the object is null and if it's an instance of a EJBLocalObject. In that case, it checks if the container of that object is the same of the object in which the assertValidLocalObject method is called. If the check don't passes, an error message is built and an exception is thrown.

Scope : The scope of this method is to check if the object passed as parameter is a Local object and belong to this container. It is used prevalently for the bean persistence management.

3 Checklist and Issues

In this section the checklist analysis will be presented.
The first section will present the checklist which have been used, then for each method it will be presented the result of the inspection. At last, the checklist point that are relative to the whole class or file will be presented.
The reader will therefore find the missing checklist points in the "Class Inspection" section, unless an error occurs in the method that is been analysed.

3.1 Assigment Checklist

Category	Number	Description
Naming Conventions	1	All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.
	2	If one-character variables are used, they are used only for temporary throwaway variables, such as those used in for loops.
	3	Class names are nouns, in mixed case, with the first letter of each word in capitalized. Examples: class Raster; class ImageSprite;
	4	Interface names should be capitalized like classes.
	5	Method names should be verbs, with the first letter of each addition word capitalized. Examples: getBackground(); computeTemperature().
	6	Class variables, also called attributes, are mixed case, but might begin with an underscore (_) followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized. Examples: _windowHeight, timeSeriesData.
	7	Constants are declared using all uppercase with words separated by an underscore. Examples: MIN_WIDTH; MAX_HEIGHT;

Indention	8	Three or four spaces are used for indentation and done so consistently
	9	No tabs are used to indent
Braces	10	Consistent bracing style is used, either the preferred Allman style (first brace goes underneath the opening block) or the Kernighan and Ritchie style (first brace is on the same line of the instruction that opens the new block).
	11	All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.
File Organization	12	Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).
	13	Where practical, line length does not exceed 80 characters.
	14	When line length must exceed 80 characters, it does NOT exceed 120 characters.
Wrapping Lines	15	Line break occurs after a comma or an operator.
	16	Higher-level breaks are used.
	17	A new statement is aligned with the beginning of the expression at the same level as the previous line.
Comments	18	Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.
	19	Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.

Java Source Files	20	Each Java source file contains a single public class or interface.
	21	The public class is the first class or interface in the file.
	22	Check that the external program interfaces are implemented consistently with what is described in the javadoc.
	23	Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).
Package and Import Statements	24	If any package statements are needed, they should be the first non- comment statements. Import statements follow.
Class and Interface Declaration	25	The class or interface declarations structure
	26	Methods are grouped by functionality rather than by scope or accessibility.
	27	Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.
Initialization and Declaration	28	Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected)
	29	Check that variables are declared in the proper scope
	30	Check that constructors are called when a new object is desired
	31	Check that all object references are initialized before use
	32	Variables are initialized where they are declared, unless dependent upon a computation
	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.

Method Calls	34	Check that parameters are presented in the correct order
	35	Check that the correct method is being called, or should it be a different method with a similar name
	36	Check that method returned values are used properly
Arrays	37	Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index)
	38	Check that all array (or other collection) indexes have been prevented from going out-of-bounds
	39	Check that constructors are called when a new array item is desired
Object Comparison	40	Check that all objects (including Strings) are compared with "equals" and not with "=="
Output Format	41	Check that displayed output is free of spelling and grammatical errors
	42	Check that error messages are comprehensive and provide guidance as to how to correct the problem
	43	Check that the output is formatted correctly in terms of line stepping and spacing

Computation, Comparisons and Assignments	44	Check that the implementation avoids brutish programming
	45	Check order of computation/evaluation, operator precedence and parenthesizing
	46	Check the liberal use of parenthesis is used to avoid operator precedence problems.
	47	Check that all denominators of a division are prevented from being zero
	48	Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding
	49	Check that the comparison and Boolean operators are correct
	50	Check throw-catch expressions, and check that the error condition is actually legitimate
	51	Check that the code is free of any implicit type conversions
Exceptions	52	Check that the relevant exceptions are caught
	53	Check that the appropriate action are taken for each catch block
Flow of Control	54	In a switch statement, check that all cases are addressed by break or return
	55	Check that all switch statements have a default branch
	56	Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions
Files	57	Check that all files are properly declared and opened
	58	Check that all files are closed properly, even in the case of an error
	59	Check that EOF conditions are detected and handled correctly
	60	Check that all file exceptions are caught and dealt with accordingly

3.2 Method : mapLocal3xException

3.2.1 Code

```
2337    private Throwable mapLocal3xException(Throwable t) {
2338
2339        Throwable mappedException = null;
2340
2341        if( t instanceof TransactionRolledbackLocalException ) {
2342            mappedException = new EJBTransactionRolledbackException();
2343            mappedException.initCause(t);
2344        } else if( t instanceof TransactionRequiredLocalException ) {
2345            mappedException = new EJBTransactionRequiredException();
2346            mappedException.initCause(t);
2347        } else if( t instanceof NoSuchObjectLocalException ) {
2348            mappedException = new NoSuchEJBException();
2349            mappedException.initCause(t);
2350        } else if( t instanceof AccessLocalException ) {
2351            mappedException = new EJBAccessException();
2352            mappedException.initCause(t);
2353        }
2354
2355        return (mappedException != null) ? mappedException : t;
2356
2357    }
2358
```

3.2.2 Checklist

Name		mapLocal3xException
Naming Conventions	**1**	Passed
	2	#2337 t is used in a "if", it isn't used for temporary throwaway variables
	3	Passed
	4	Passed
	5	See Class Inspection Section
	6	Passed
	7	See Class Inspection Section
Indention	**8**	Passed
	9	Passed
Braces	**10**	Passed, Kernighan and Ritchie style
	11	Passed
Wrapping Lines	**15**	Passed
	16	Passed

	17	Passed
Comments	**18**	Passed, no comments
	19	Passed, no comments
Java Source Files	**20**	Passed
	21	Passed
	22	Passed
	23	Incomplete Javadoc .
Method Calls	**34**	Passed
	35	Passed
	36	Passed
Arrays	**37**	Passed, no arrays
	38	Passed, no arrays
	39	Passed, no arrays
Object Comparison	**40**	Passed
Output Format	**41**	Passed
	42	Passed
	43	Passed
Computation, Comparisons and Assignments	**44**	Passed
	45	Passed
	46	Passed
	47	Passed
	48	Passed
	49	Passed
	50	Passed
	51	Passed
Exceptions	**52**	Passed
	53	Passed
Flow of Control	**54**	Passed, no switches
	55	Passed,no switches
	56	Passed,no loops
Files	**57**	Passed
	58	Passed
	59	Passed
	60	Passed

3.3 Method : authorize

3.3.1 Code

```
2359    /**
2360     * Common code to handle EJB security manager authorization call.
2361     */
2362    public boolean authorize(EjbInvocation inv) {
2363
2364        // There are a few paths (e.g. authorizeLocalMethod,
2365        // authorizeRemoteMethod, Ejb endpoint pre-handler )
2366        // for which invocationInfo is not set.  We get better
2367        // performance with the security manager on subsequent
2368        // invocations of the same method if invocationInfo is
2369        // set on the invocation.  However, the authorization
2370        // does not depend on it being set.  So, try to set
2371        // invocationInfo but in this case don't treat it as
2372        // an error if it's not available.
2373        if( inv.invocationInfo == null ) {
2374
2375            inv.invocationInfo = getInvocationInfo(inv);
2376
2377        }
2378
2379        // Internal methods for 3.0 bean creation so there won't
2380        // be corresponding permissions in the security policy file.
2381        if( (inv.method.getDeclaringClass() == localBusinessHomeIntf)
2382            ||
2383            (inv.method.getDeclaringClass() == remoteBusinessHomeIntf) ) {
2384            return true;
2385        }
2386
2387        boolean authorized = securityManager.authorize(inv);
2388
2389        if( !authorized ) {
2390
2391            if( inv.context != null ) {
2392                // This means that an enterprise bean context was created
2393                // during the authorization call because of a callback from
2394                // a JACC enterprise bean handler. Since the invocation will
2395                // not proceed due to the authorization failure, we need
2396                // to release the enterprise bean context.
2397                releaseContext(inv);
2398            }
2399        }
2400
2401        return authorized;
2402    }
```

3.3.2 Checklist

Name		authorize
Naming Conventions	**1**	Passed
	2	Passed
	3	Passed
	4	Passed
	5	See Class Inspection Section
	6	Passed
	7	See Class Inspection Section
Indention	**8**	#2374-2376 inconsistent spacing, #2382 exceeding space
	9	Passed
Braces	**10**	Passed, Kernighan and Ritchie style
	11	Passed
Wrapping Lines	**15**	#2382 exceeding newline before !!
	16	Passed
	17	Passed
Comments	**18**	Passed
	19	Passed
Java Source Files	**20**	Passed
	21	Passed

	22	Passed, implementation of the "Container" interface method
	23	Passed, present both in BaseContainer and Container Javadoc
Initialization and Declarations	**28**	Passed
	33	Misplaced declaration at row #2387
Method Calls	**34**	Passed
	35	Passed
	36	Passed
Arrays	**37**	Passed,no array
	38	Passed,no array
	39	Passed,no array
Object Comparison	**40**	#2373 "==" is used instead of "equals",#2381, #2383 "==" is used instead of "equals"
Output Format	**41**	Passed
	42	Passed
	43	Passed

Computation, Comparisons and Assignments	**44**	Passed
	45	Passed
	46	Passed
	47	Passed
	48	Passed
	49	Passed
	50	Passed
	51	Passed
Exceptions	**52**	Passed
	53	Passed
Flow of Control	**54**	Passed,no switch
	55	Passed,no switch
	56	Passed,no loop
Files	**57**	Passed
	58	Passed
	59	Passed
	60	Passed

3.4 Method : initializeEjbInterfaceMethods

3.4.1 Code

```
2404    /**
2405     * Create an array of all methods in the standard EJB interfaces:
2406     * javax.ejb.EJB(Local){Home|Object} .
2407     */
2408    private void initializeEjbInterfaceMethods()
2409        throws Exception
2410    {
2411        ejbIntfMethods = new Method[EJB_INTF_METHODS_LENGTH];
2412
2413        if ( isRemote ) {
2414            ejbIntfMethods[ EJBHome_remove_Handle ] =
2415                EJBHome.class.getMethod("remove",
2416                                        new Class[]{javax.ejb.Handle.class});
2417            ejbIntfMethods[ EJBHome_remove_Pkey ] =
2418                EJBHome.class.getMethod("remove",
2419                                            new Class[]{java.lang.Object.class});
2420            ejbIntfMethods[ EJBHome_getEJBMetaData ] =
2421                EJBHome.class.getMethod("getEJBMetaData", NO_PARAMS);
2422            ejbIntfMethods[ EJBHome_getHomeHandle ] =
2423                EJBHome.class.getMethod("getHomeHandle", NO_PARAMS);
2424
2425            ejbIntfMethods[ EJBObject_getEJBHome ] =
2426                EJBObject.class.getMethod("getEJBHome", NO_PARAMS);
2427            ejbIntfMethods[ EJBObject_getPrimaryKey ] =
2428                EJBObject.class.getMethod("getPrimaryKey", NO_PARAMS);
2429            ejbIntfMethods[ EJBObject_remove ] =
2430                EJBObject.class.getMethod("remove", NO_PARAMS);
2431            ejbIntfMethods[ EJBObject_getHandle ] =
2432                EJBObject.class.getMethod("getHandle", NO_PARAMS);
2433            ejbIntfMethods[ EJBObject_isIdentical ] =
2434                EJBObject.class.getMethod("isIdentical",
2435            new Class[]{javax.ejb.EJBObject.class});
2436
2437            if ( isStatelessSession ) {
2438                if( hasRemoteHomeView ) {
2439                    ejbIntfMethods[ EJBHome_create ] =
2440                        homeIntf.getMethod("create", NO_PARAMS);
2441                }
2442            }
2443        }
2444
```

```
2444
2445        if ( isLocal ) {
2446            ejbIntfMethods[ EJBLocalHome_remove_Pkey ] =
2447                EJBLocalHome.class.getMethod("remove",
2448                    new Class[]{java.lang.Object.class});
2449
2450            ejbIntfMethods[ EJBLocalObject_getEJBLocalHome ] =
2451                EJBLocalObject.class.getMethod("getEJBLocalHome", NO_PARAMS);
2452            ejbIntfMethods[ EJBLocalObject_getPrimaryKey ] =
2453                EJBLocalObject.class.getMethod("getPrimaryKey", NO_PARAMS);
2454            ejbIntfMethods[ EJBLocalObject_remove ] =
2455                EJBLocalObject.class.getMethod("remove", NO_PARAMS);
2456            ejbIntfMethods[ EJBLocalObject_isIdentical ] =
2457                EJBLocalObject.class.getMethod("isIdentical",
2458            new Class[]{javax.ejb.EJBLocalObject.class});
2459
2460            if ( isStatelessSession ) {
2461                if( hasLocalHomeView ) {
2462                    Method m = localHomeIntf.getMethod("create", NO_PARAMS);
2463                    ejbIntfMethods[ EJBLocalHome_create ] = m;
2464                }
2465            }
2466        }
2467
2468    }
2469
```

3.4.2 Checklist

Name		initializeEjbInterfaceMethods
Naming Conventions	**1**	Passed
	2	#2462 m is defined in a if.
	3	Passed
	4	Passed
	5	See Class Inspection Section
	6	Passed
	7	See Class Inspection Section
Indention	**8**	#2416 exceeding space, #2419 exceeding space, #2424 inconsistent space, #2435 indentation incorrect, #2449 incosistent space, #2458 indentation incorrect.
	9	Passed
Braces	**10**	Kernighan and Ritchie style, #2408 Allman style.
	11	Passed
File Organization	**12**	Exceeding Blank line: #2424, #2449
	13	Passed (max = 78)
	14	Passed
Wrapping Lines	**15**	Passed
	16	#2416, #2448, #2458, #2435 are incorrect
	17	Passed
Comments	**18**	Passed
	19	Passed
Java Source Files	**20**	Passed

	21	Passed
	22	Passed
	23	Not passed: the javadoc doesn't cover this method.
Initialization and Declarations	**31**	#2437,#2460 isStatelessSession can lead to an error.
	32	Passed
	33	Misplaced declaration at row #2462
Method Calls	**34**	Passed
	35	Passed
	36	Passed
Arrays	**37**	Passed
	38	Passed
	39	Passed
Object Comparison	**40**	Passed
Output Format	**41**	Passed
	42	Passed,no error message
	43	Passed
Computation, Comparisons and Assignments	**44**	Passed
	45	Passed
	46	Passed
	47	Passed

	48	Passed
	49	Passed
	50	Passed
	51	Passed
Exceptions	**52**	Passed
	53	Passed
Flow of Control	**54**	Passed, no switch
	55	Passed, no switch
	56	Passed, no loop
Files	**57**	Passed, no file
	58	Passed, no file
	59	Passed, no file
	60	Passed, no file

3.5 Method : getJaccEjb

3.5.1 Code

```
2676    public Object getJaccEjb(EjbInvocation inv) {
2677        Object bean = null;
2678
2679        // Access to an enterprise bean instance is undefined for
2680        // anything but business method invocations through
2681        // Remote , Local, and ServiceEndpoint interfaces.
2682        if( ( (inv.invocationInfo != null) &&
2683              inv.invocationInfo.isBusinessMethod )
2684            ||
2685            inv.isWebService ) {
2686
2687            // In the typical case the context will not have been
2688            // set when the policy provider invokes this callback.
2689            // There are some cases where it is ok for it to have been
2690            // set, e.g. if the policy provider invokes the callback
2691            // twice within the same authorization decision.
2692            if( inv.context == null ) {
2693
2694                try {
2695                    inv.context = getContext(inv);
2696                    bean = inv.context.getEJB();
2697                    // NOTE : inv.ejb is not set here.  Post-invoke logic for
2698                    // BaseContainer and webservices uses the fact that
2699                    // inv.ejb is non-null as an indication that that
2700                    // BaseContainer.preInvoke() proceeded past a certain
2701                    // point, which affects which cleanup needs to be
2702                    // performed.  It would be better to have explicit
2703                    // state in the invocation that says which cleanup
2704                    // steps are necessary(e.g. for invocationMgr.postInvoke
2705                    // , postInvokeTx, etc) but I'm keeping the logic the
2706                    // same for now.   BaseContainer.authorize() will
2707                    // explicitly handle the case where a context was
2708                    // created as a result of this call and the
2709                    // authorization failed, which means the context needs
2710                    // be released.
2711
2712                } catch(EJBException e) {
2713                    _logger.log(Level.WARNING, CONTEXT_FAILURE_JACC, logParams[0]);
2714                    _logger.log(Level.WARNING, "", e);
2715                }
2716
2717            } else {
2718                bean = inv.context.getEJB();
2719            }
2720        }
2721
2722        return bean;
2723    }
```

3.5.2 Checklist

Name		getJaccEjb
Naming Conventions	**1**	Passed
	2	Passed
	3	Passed
	4	Passed
	5	See Class Inspection Section
	6	Passed
	7	See Class Inspection Section
Indention	**8**	#2683 exceeding space
	9	Passed for this method, but found 170 occurrences in other functions
Braces	**10**	Passed for the methods (Kernighan and Ritchie style), but not consistent for all the file
	11	Passed
Wrapping Lines	**15**	#2684 exceeding newline before "! !".
	16	Passed
	17	Passed
Comments	**18**	Passed
	19	Passed
Java Source Files	**20**	Passed
	21	Passed
	22	Passed, implementation of the "Container" interface method
	23	Passed, present both in BaseC-ontainer and Container javadoc

Method Calls	**34**	Passed
	35	Passed
	36	Passed
Arrays	**37**	Passed, no arrays
	38	Passed, no arrays
	39	Passed, no arrays
Object Comparison	**40**	Passed
Output Format	**41**	Passed
	42	Passed
	43	Passed
Computation, Comparisons and Assignments	**44**	Passed
	45	Passed
	46	Passed
	47	Passed
	48	Passed
	49	Passed
	50	Passed
	51	Passed
Exceptions	**52**	Passed
	53	Passed
Flow of Control	**54**	Passed
	55	Passed
	56	Passed
Files	**57**	Passed
	58	Passed
	59	Passed
	60	Passed

3.6 Method : assertValidLocalObject

3.6.1 Code

```
2725    public void assertValidLocalObject(Object o) throws EJBException
2726    {
2727        boolean valid = false;
2728        String errorMsg = "";
2729
2730        if( (o != null) && (o instanceof EJBLocalObject) ) {
2731            // Given object is always the client view EJBLocalObject.
2732            // Use utility method to translate it to EJBLocalObjectImpl
2733            // so we handle both the generated and proxy case.
2734            EJBLocalObjectImpl ejbLocalObjImpl =
2735                EJBLocalObjectImpl.toEJBLocalObjectImpl( (EJBLocalObject) o);
2736            BaseContainer otherContainer =
2737                (BaseContainer) ejbLocalObjImpl.getContainer();
2738            if( otherContainer.getContainerId() == getContainerId() ) {
2739                valid = true;
2740            } else {
2741                errorMsg = "Local objects of ejb-name " + otherContainer.ejbDescriptor.getName() +
2742                        " and ejb-name " + ejbDescriptor.getName() +
2743                         " are from different containers" ;
2744
2745            }
2746        } else {
2747            errorMsg = (o != null) ?
2748                    "Parameter instance of class '" +  o.getClass().getName() +
2749                            "' is not a valid local interface instance for bean " +
2750                                    ejbDescriptor.getName()
2751                  :
2752             "A null parameter is not a valid local interface of bean " +  ejbDescriptor.getName();
2753        }
2754
2755        if( !valid ) {
2756            throw new EJBException(errorMsg);
2757        }
2758
2759    }
```

3.6.2 Checklist

Name		assertValidLocalObject
Naming Conventions	**1**	Passed
	2	#2725: The parameter "o" should be called "object" since it's not a throwaway variable
	3	Passed
	4	Passed
	5	See Class Inspection Section
	6	Passed
	7	See Class Inspection Section
Indention	**8**	#2743 exceeding space, #2749-2752 inconsistent spacing
	9	Passed
Braces	**10**	Passed for the methods (Kernighan and Ritchie style), but not consistent for all the file
	11	Passed
Wrapping Lines	**15**	Passed
	16	Passed
	17	Passed
Comments	**18**	Passed
	19	Passed
Java Source Files	**20**	Passed
	21	Passed
	22	Passed, implementation of the "Container" interface method
	23	Passed, present both in BaseContainer and Container javadoc

Method Calls	**34**	Passed
	35	Passed
	36	Passed
Arrays	**37**	Passed, no Arrays
	38	Passed, no Arrays
	39	Passed, no Arrays
Object Comparison	**40**	#2738 "==" used instead of .equals for comparing two non-null objects
Output Format	**41**	Passed
	42	Passed
	43	Passed
Computation, Comparisons and Assignments	**44**	Passed
	45	Passed
	46	Passed
	47	Passed
	48	Passed
	49	Passed
	50	Passed
	51	Passed
Exceptions	**52**	Passed
	53	Passed
Flow of Control	**54**	Passed
	55	Passed
	56	Passed
Files	**57**	Passed
	58	Passed
	59	Passed
	60	Passed

4 Other problems

Although no particular problems have been identified during the methods inspection, some problems within the assigned class have been found, as highlighted in the previous section.

4.1 Class Inspection

This section contains the checklist that are general for the class or the file and not only to the methods that were assigned to us. It also contains method-specific problems that have been found in other methods belonging to the **BaseContainer** class.
Note that, unless a specific error is noted, the "passed" statement is referred

to the declaration part of the class and not to the other methods' body, since the whole class have been checked only when an automated process has been used.

4.1.1 Checklist

Indention	**9**	Found **170** occurrences in the analysed file
	5	These methods: #1836 externalPreInvoke(), #1866 externalPostInvoke(), #1922 preInvoke(), #2034 webServicePostInvoke(), #1731 _constructEJBContextImpl(), #1736 _constructEJBInstance(), don't respect the convention.
	7	Methods #1836 externalPreInvoke(), #1866 externalPostInvoke(), #1922 preInvoke(), #2034 webServicePostInvoke(), #1731 _constructEJBContextImpl(), #1736 _constructEJBInstance(), don't respect the convention.
Braces	**10**	Kernighan and Ritchie style is prevalent but not consistent for all the file
File Organization	**12**	#564-562 are not consistent with the file style
	13	#207-243 exceeds 80 characters several time. Log messages could be written in several lines. #268, #273, inline comments exceeding optimal row length
	14	#2101 Line too long, #2249 string too long, #1628 line too long, #4824, #4880, line too long

Package and Import Statements	**24**	Passed
Class and Interface Declarations	**25**	#189-582 Variables are not declared in the right order
	26	Passed
	27	Passed
Initialization and Declarations	**28**	Passed
	29	Passed
	30	Passed
	31	Passed
	32	Passed
	33	Passed

5 References

- Assignment document part 3: Document structure and checklist
- "Brutish Programming", Dr. John Dalbey: Code quality inspection
- http://glassfish.pompel.me/ Javadoc for Glassfish
- http://www.javadocumentation.com/ Javadoc for the specific package we analysed
- Enterprise JavaBeansTM Specification Version 2.0 - Sun Microsystems
- Oracle Documentation: Component descriptions and some JavaDocs

6 Tools

- **SVN**: For the checkout of the source code
- **Gedit**: Text editor for code inspection
- **GrepCode**: Preliminary package analysis
- **grep**: For usage inspection
- **TexStudio**: Document editor

7 Work Hours

- Mattia Fontana: 34 hours
- Edoardo Giacomello: 36,5 hours