Methods:

* **Name:**

setEJBObjectTargetMethodInfo( InvocationInfo invInfo , boolean isLocal , Class originalIntf ) 3332-3386

**Location:**

appserver/ejb/ejb-container/src/main/java/com/sun/ejb/containers/BaseContainer.java

* **Name:**

addLocalRemoteInvocationInfo( ) 3425-3552

**Location:**

appserver/ejb/ejb-container/src/main/java/com/sun/ejb/containers/BaseContainer.java

* **Name:**

addWSOrTimedObjectInvocationInfo( ) 3553-3674

**Location:**

appserver/ejb/ejb-container/src/main/java/com/sun/ejb/containers/BaseContainer.java

**//ejbIntfClazz is not meaningful (1)**

**//m solo un carattere no va bene (2)**

**//(3) OK, c’è una sola classe, che è quella del file, e ha la maiuscola.**

**//(4) se è vero che l’unica classe dichiarata è quella del file, le uniche interfacce Container, EjbContainerFacade, JavaEEContainer sono OK.**

**//(5) i nomi dei metodi son tutti verbi e hanno tutte le parole interne con la maiuscola.**

**//(7) di Basecontainer le costanti dichiarate dalla riga 262 alla 277 non hanno tutti i caratteri in uppercase.**

**//8 e 9,10,11 OK**

**//12 VA BENE, 13 RIGA 3353 84 CARATTERI (OK LA COSTANTE HA UN NOME LUNGO) 14 OK.**

**//17 va bene, 15 e 16 chiedere**

**//18 manca la descrizione generale di cosa fa il metodo (nonostante ci siano commenti all interno)**

**//19 OK, 20 RIGA 4998 E 5010 DUE CLASSI PUBBLICHE , 21 OK**

**//22 E 23 CHIEDERE, 24 OK.**

**//25 PUNTO D ED E, DICIAMO CHE NON è RISPETTATO L ORDINE DI PUBLIC , PROTECTED E PRIVATE PERCHè SI ALTERNANO.**

**//588 riga costruttore 25 F E G OK. 26 E 27 VA RIGUARDARE (IMPOS)**

**//28 E 29 CHIEDERE E DA 30 A 33 OK, 34,35,36 DA FARE**

private void setEJBObjectTargetMethodInfo(InvocationInfo invInfo, **//non è comment il metodo (18)**

boolean isLocal,

Class originalIntf)

throws EJBException {

Class ejbIntfClazz = isLocal ? **//ejbIntfClazz is not meaningful (1)**

javax.ejb.EJBLocalObject.class : javax.ejb.EJBObject.class; **//?????**

Class[] paramTypes = invInfo.method.getParameterTypes();

String methodName = invInfo.method.getName();

// Check for 2.x Remote/Local bean attempts to override

// EJBObject/EJBLocalObject operations.

if( ejbIntfClazz.isAssignableFrom(originalIntf) ) {

try {

Method m = ejbIntfClazz.getMethod(methodName, paramTypes); **//m solo un carattere no va bene!**

// Attempt to override EJBObject/EJBLocalObject method. Print

// warning but don't treat it as a fatal error. At runtime, the

// EJBObject/EJBLocalObject method will be called.

String[] params = { m.toString(),invInfo.method.toString() };

\_logger.log(Level.WARNING, ILLEGAL\_EJB\_INTERFACE\_OVERRIDE, params);

invInfo.ejbIntfOverride = true;

return;

} catch(NoSuchMethodException nsme) {

}

}

try {

invInfo.targetMethod1 = ejbClass.getMethod(methodName, paramTypes);

if( isSession && isStatefulSession ) {

MethodDescriptor methodDesc = new MethodDescriptor **(controllare se si può andar a capo)**

(invInfo.targetMethod1, MethodDescriptor.EJB\_BEAN);

// Assign removal info to inv info. If this method is not

// an @Remove method, result will be null.

invInfo.removalInfo = ((EjbSessionDescriptor)ejbDescriptor).

getRemovalInfo(methodDesc);

}

} catch(NoSuchMethodException nsme) {

Object[] params = { logParams[0] + ":" + nsme.toString(),

(isLocal ? "Local" : "Remote"),

invInfo.method.toString() };

\_logger.log(Level.WARNING, BEAN\_CLASS\_METHOD\_NOT\_FOUND, params);

// Treat this as a warning instead of a fatal error.

// That matches the behavior of the generated code.

// Mark the target methods as null. If this method is

// invoked at runtime it will be result in an exception from

// the invocation handlers.

invInfo.targetMethod1 = null;

}

}

ambiguity:

1. Check number 1: do we only have to check names in our methods or in our whole file?