POLITECNICO DI MILANO

Master Degree course in Computer Science Engineering

Third Assignment: Code Inspection

Instructor: Prof. Elisabetta Di Nitto

Authors:

Brunato Andrea	851413
Costantini Lorenzo	852599
Dell'Orto Alessandro	853050

TABLE OF CONTENT:

1. Classes assigned to the group:

- ConnectorsUtil in the following location:

`appserver/connectors/connectors-internal-api/src/main/java/com/sun/appserv/connectors/internal/api`

- AppSpecificConnectorClassLoaderUtil in the following location:

`appserver/connectors/connectors-internal-api/src/main/java/com/sun/appserv/connectors/internal/api`

2. Functional role of assigned set of classes:

“In GlassFish, Connectors are the preferred integration mechanism for all application server provided services like JMS/JDBC/JAXR access etc. The primary design philosophy is to provide a unified connector based architecture for all AS provided services. From a user's point of view, resources would continue to be shown as JDBC or JMS resources, but behind-the-scenes all these resources are connector resources. This plumbing of application server provided services like JDBC/JMS to connectors is achieved via System resource adapters. These system resource adapters are pre-installed/pre-configured and lazily loaded on first resource access. As explained in the Connection Pooling page, there is a single connection pool infrastructure for JDBC/JMS/EIS resources, thanks to this connector-centric architecture.”

This quote specifies what a connector is and what does the resource adapter, and it is taken from:
https://glassfish.java.net/javaee5/integration-tech/glassfish_connectors.html.

Those two concepts are fundamental and recurrent in our two classes. Often resource adapters are indicated as RA in the code (JMS-RA, JDBC-RA..). Instead the acronym EAR stands for Embedded Resource Adapter, that are related to connector modules deployed as a Java EE component in a Java EE application. Such connectors are only visible to components residing in the same Java EE application.

ConnectorsUtil

ConnectorsUtil is a static class providing utility methods to manage various aspects and functionalities related to Connectors, as synthetically reported in the documentation. Many of these methods deal with RAR files (RAR stands for Resource Adapter Archive, where a resource adapter is stored), and the method at line 1020 let us know what defines a RAR resource in the ConnectorsUtil class.

```
1020    public static boolean isRARResource(Resource resource){
1021        return resource instanceof ConnectorResource ||
1022                resource instanceof AdminObjectResource ||
1023                resource instanceof ConnectorConnectionPool ||
1024                resource instanceof ResourceAdapterConfig ||
1025                resource instanceof WorkSecurityMap;
1026    }
1027
```

These five classes names corresponds to 5 of the 10 subelements of the element “resources” of the domain.xml file that contains most of the Oracle GlassFish Server configuration. Here at this link it can be seen its complete element hierarchy:
https://docs.oracle.com/cd/E19798-01/821-1753/abhaq/index.html.

One of the most important methods offered by this class is **extractRar(String fileName, String rarName, String destDir)** that allows to extract the .rar from the uber jar into specified directory.
Then this class exposes several getter methods that return names of resources, modules, pools, applications, RAR and embedded RARs, that presumably will be called by other classes in order to find those objects in the right files and directories:

- **public static String getLocation(String moduleName)** returns a string containing the location of a RAR belonging to the system or embedded into an application, by calling the next three methods.
- **public static String getSystemModuleLocation(String moduleName)** returns the name of the installation directory of system RARs, and the module name is the RAR name, all according to the method's javadoc.
- **public static String getRarNameFromApplication(String appName)** returns the embedded Rar name given the name of the application that contains the Rar.
- **public static String getApplicationNameOfEmbeddedRar(String embeddedRarName)** does the opposite of the previous method as the name of the method suggests.
- **public static String getEmbeddedRarModuleName(String applicationName, String moduleName)** returns a concatenation of strings containing the application name and the module name.
- **public static String getRarNameOfResource(Resource resource, Resources resources)** returns the resource adapter name given a Rar Resource, that is an object instance of one of the classes described in the method isRARResource.
- **public static List<WorkSecurityMap> getWorkSecurityMaps(String raName, Resources allResources)** is quite the opposite of the previous method, but only for the RAR resource of the type WorkSecurityMap. Given the resource adapter name, it "returns all the configured connector-work-security-maps for the .rar", as written in the Javadoc. There are other methods that we won't mention since this class is really big that retrieves objects given names.
- **public static String getResourceAdapterNameOfPool(String poolName, Resources allResources)** returns the resource adapter name given the connection pool name.
- **getApplicationName(DeploymentContext context), getApplicationName(Resource resource), getModuleName(EjbDescriptor descriptor),** are others methods dealing

with the retrieving of strings by informations about objects or other strings passed as a parameter.

In this class a certain number of methods refer to the Java Naming and Directory Interface (JNDI), which is a Java API for a directory service that allows Java software clients to discover and look up data and objects via a name.
public static String getReservePrefixedJNDINameForResource(String compId, String resourceName, JavaEEResourceType resType) is the most interesting method regarding JNDI; given all the possible JavaEEResourceTypes, different prefixes are associated to the resourceName string, and the method returns a string that concatenates two prefixes, the resource name and also the string compId if the latter is not null.

Other methods such as **isStandAloneRA(String moduleName), belongsToSystemRA (String raName)**, **belongsToJdbcRA(String raName), isJMSRA(String moduleName)** check whether a certain resource adapter is belonging to the system or not.

Other methods such as **public static Set getMergedActivationConfigProperties (EjbMessageBeanDescriptor msgDesc), public static boolean isDynamicReconfigurationEnabled(ResourcePool pool),**
public static ResourceAdapterConfig getRAConfig(String raName, Resources allResources) deals with configurations.

Then there are functions dealing with connection pools as:

- **public static Collection<ConnectorConnectionPool> getAllPoolsOfModule(String moduleName, Resources allResources)** that given the resource adapter name returns a list of connection pools.
- **public static Collection<Resource> getAllSystemRAResourcesAndPools(Resources allResources)** returns all the system RAR resources and pools.

- **public static Collection<String> getAllPoolNames(Collection<ConnectorConnectionPool> connectionPools)** returns a collection of pool names (strings).

Finally other methods are present in this class that deals with topic unrelated to the ones presented before, for example **public static long getShutdownTimeout (ConnectorService connectorService)** whose function is expressed in the javadoc later in the document.

AppSpecificConnectorClassLoaderUtil

A Class Loader is an object that is responsible for loading classes, according to the Java 7 documentation.
The Oracle GlassFish Server Application Development Guide contains a section dedicated to the class loader hierarchy.
The guide is visible at the following link where the presented quotes are written:
https://docs.oracle.com/cd/E18930_01/html/821-2418/beadf.html
Two class loaders among all the others are important to understand the reason why this class exists and the functionalities that provides.
The Connector Class Loader “is a single class loader instance that loads individually deployed connector modules, which are shared across all applications. It is parent to the Applib class loader and the LifeCycleModule class loader”.
The Applib Class Loader “loads the library classes, specified during deployment, for a specific enabled module or Java EE application.”
AppSpecificConnectorClassLoaderUtil is a util class that, as expressed in its name, contains several support methods to deal with Application-Specific class loading.
This class has several private attributes:

- The logger class used to log messages for a specific system or application component
- Several instances of the Provider class which represents a "provider" for the Java Security API, where a provider implements some or all parts of Java Security.

Again, also this class deals with RAR and the contained RA:
detectReferredRARs(String appName) is a very complex method that retrieves resource adapters references given the application name, searching through all bundle descriptors, ejb-descriptors and managed-bean descriptors. Comment lines from 135 to 146 point out the goals of this method, that will be implemented in the various following private methods.
public boolean useGlobalConnectorClassLoader() controls if the class loading policy is global or not. According to the Oracle GlassFish Server 3.0.1 Application Development Guide, class loading policy can be "derived" or "global" and determines which resource adapters are accessible to applications:

- **derived** means that applications access resource adapters are based on references in their deployment descriptors. These references can be *resource-ref, resource-env-ref, resource-adapter-mid*, or equivalent annotations.
- **global** means that all stand-alone resource adapters are available to all applications.

getRARsReferredByApplication(String appName) returns a set of strings representing the Rar referred by the application, whose string appName is passed as a parameter
getRequiredResourceAdapters(String appName) retrieves the required resource adapters of an application indicated by the string appName.

3. List of issues found by applying the checklist:

Index of methods assigned:

Class AppSpecificConnectorClassLoaderUtil

1)Name:getRequiredResourceAdapters(String appName)
Start Line:464
Location:
appserver/connectors/connectors-internal-api/src/main/java/com/sun/appserv/connectors/internal/api/AppSpecificConnectorClassLoaderUtil.java

Class ConnectorsUtil

1)Name:getLocation(String moduleName)
Start Line:163
Location:
appserver/connectors/connectors-internal-api/src/main/java/com/sun/appserv/connectors/internal/api/ConnectorsUtil.java
2)Name:getAllSystemRAResourcesAndPools(Resources allResources)
Start Line:330
Location:appserver/connectors/connectors-internal-api/src/main/java/com/sun/appserv/connectors/internal/api/ConnectorsUtil.java
3)Name:isDynamicReconfigurationEnabled(ResourcePool pool)
Start Line:408
Location:appserver/connectors/connectors-internal-api/src/main/java/com/sun/appserv/connectors/internal/api/ConnectorsUtil.java
4)Name:
getMergedActivationConfigProperties(EjbMessageBeanDescriptor msgDesc)
Start Line:435
Location:appserver/connectors/connectors-internal-api/src/main/java/com/sun/appserv/connectors/internal/api/ConnectorsUtil.java
5)Name:getShutdownTimeout(ConnectorService connectorService)
Start Line:483

Location:appserver/connectors/connectors-internal-api/src/main/java/com/sun/appserv/connectors/internal/api/ConnectorsUtil.java
6)Name:getTransactionIsolationInt(int tranIsolation)
Start Line:537
Location:appserver/connectors/connectors-internal-api/src/main/java/com/sun/appserv/connectors/internal/api/ConnectorsUtil.java
8)Name:getReservePrefixedJNDINameForResource(String compId , String resourceName , JavaEEResourceType resType)
Start Line:561
Location:appserver/connectors/connectors-internal-api/src/main/java/com/sun/appserv/connectors/internal/api/ConnectorsUtil.java

3.1 General issues of AppSpecificConnectorClassLoaderUtil class

Issue 1: The name of the class is not totally clear, and this problem becomes more important because of the lack of the javadoc associated to the class who could have explained the meaning of words “AppSpecific” for example. After reading the Oracle documentation the meaning of the class is understandable, but something should have been written on the Javadoc.

Issue 23: Almost every method of the class lacks a Javadoc.
Only two methods of the whole class AppSpecificConnectorClassLoaderUtil have a Javadoc inherited thanks to the {@inheritDoc}command. This is one of the most important problems found in this part of code and especially in this class.

Issue 25: The class variable _logger is declared after all the other private instance variables; the right order of declaration should be the opposite.

Issue 26: Methods of the class AppSpecificConnectorClassLoaderUtil seem to be grouped a bit casually and not by functionality. This class consists for the most part by big private methods called directly by “public void detectReferredRARs(String appName)” or by one of the private methods correlated to that public method at a deeper level of call. Those private methods are all subsequent, apart “Set<String>

getRARsReferredByApplication(String appName)" at line 218, which is not correlated to the complex computation that runs from line 118 to 440, and should be placed after the last private method of the functional group of "detectReferredRars(String appName)".
Methods at line 442, 450 and "getRequiredResourceAdapters(String appName)" at line 464 are all public and indipendent from each other and they are placed right after the above mentioned group of connected private methods.
At line 487, 491, 495 there are three simple private getter whose computation is one line long. They are called by the submethods of detectReferredRARs and they probably should have been placed before line 442.

Issue 27: the class AppSpecificConnectorClassLoaderUtil contains three overly long methods starting at line 252, 332 and 383. The private method "private void detectResourceInRA(Application app, String moduleName, String jndiName)" at line 252 is the most complex of those three and surely needs to be divided in more sub-private methods which can do smaller parts of its computation.

3.1.1 getRequiredResourceAdapters(String appName)

```
464    public Collection<String> getRequiredResourceAdapters(String appName) {
465        List<String> requiredRars = new ArrayList<String>();
466        if (appName != null) {
467            ConnectorService connectorService = connectorServiceProvider.get();
468            //it is possible that connector-service is not yet defined in domain.xml
469
470            if (connectorService != null) {
471                if (appName.trim().length() > 0) {
472                    Property property = connectorService.getProperty(appName.trim());
473                    if (property != null) {
474                        String requiredRarsString = property.getValue();
475                        StringTokenizer tokenizer = new StringTokenizer(requiredRarsString, ",");
476                        while (tokenizer.hasMoreTokens()) {
477                            String token = tokenizer.nextToken().trim();
478                            requiredRars.add(token);
479                        }
480                    }
481                }
482            }
483        }
484        return requiredRars;
485    }
```

This method contains no errors for what regards indention and braces style. Four spaces are used consistently for indention, and the braces style used is the “Kernighan and Ritchie” one. Tabs are never used.

Issue 23: The examined method doesn’t have a Javadoc associated to it, as almost every other method of the class AppSpecificConnectorClassLoaderUtil.

Issue 52: there should be an exception that controls the case in which the string in input “appName” is null, and returning a null object instead of a collection of string could lead to bugs.

There are no other issues found on this method:

- no files, outputs or switch statements are present.
- the while statement at line 476 is well defined and the termination and increment is handled in the hasMoreTokens() method of StringTokenizer.
- variables are always initialized when they are declared, and declarations appear always at beginning of blocks.
- no problems found about comparisons, use of constructors and assignments.

3.2 General issues of ConnectorsUtil class

Issue 19: Code has been commented out without any future planification, indeed isn't expressed a date in which it will be deleted. Certain commented lines aren't obsolete but code under construction, for instance at line 187 and 999, and some TODO tags associated to future implementation of situations not yet taken into account are present at line 968 and 969. Those four code sections are a clear signal of bugs in the current version of Glassfish.

Issue 23: The Javadoc of the class assigned is incomplete and where it's specified is usually too minimal and not much explanatory, apart the one at line 767 which provides little more informations about the system. Missing javadocs for methods at lines: 111,121,148,155,163,271,297,378,408,466,473,555,561,615,622,626, 634,644,654,659,670,711,715,724,728,739,743,752,828,879,883,888,897,901,905, 909,913,931,935,948,964,970,1006,1020,1028.
Even if the class Javadoc is present, this should be much more explicative, also given the fact that the class is really big.

Issue 26: Methods aren't grouped w.r.t. functionality but quite at random, they probably have been left with the order in which they have been created.

Issue 27: The code has too many methods and this can lead to a cohesion problem. For example some methods related to JNDI or all the methods from line 408 to 622 (including method 4 and 7 of our code inspection) are not really related to the management of RAR. In order to propose a separation of this big class in two different static classes, it's possible to move on another class the RAR related methods and maintaining on ConnectorsUtil all the methods that are involved in some controls based on the constants of ConnectorConstants interface (excluding the one that use ConnectorConstants.EMBEDDEDRAR_NAME_DELIMITER). It could be useful also to move on another class all the methods related to JNDI, which is the second cluster in order of importance, after the RAR management cluster, for what regards logical connection between methods and the relative

number of those methods with respect to the total number of the ones of ConnectorsUtil.

3.2.1 getLocation(String moduleName)

```
162
163    public static String getLocation(String moduleName) throws ConnectorRuntimeException {
164        String location = null;
165        if(ConnectorsUtil.belongsToSystemRA(moduleName)){
166            location = ConnectorsUtil.getSystemModuleLocation(moduleName);
167        }else{
168            location = internalGetLocation(moduleName);
169            if(location == null){
170                //check whether its embedded RAR
171                String rarName = getRarNameFromApplication(moduleName);
172                String appName = getApplicationNameOfEmbeddedRar(moduleName);
173
174                if(appName != null && rarName != null){
175                    location = internalGetLocation(appName);
176                    if(location != null){
177                        location = location + File.separator + rarName + "_rar";
178                    }else{
179                        throw new ConnectorRuntimeException("Unable to find location for module : " + moduleName);
180                    }
181                }
182            }
183        }
184        return location;
185        /* TODO V3
186
187            if(moduleName == null) {
188                return null;
189            }
190            String location  = null;
191            ConnectorModule connectorModule =
192                    dom.getApplications().getConnectorModuleByName(moduleName);
193            if(connectorModule != null) {
194                location = RelativePathResolver.
195                        resolvePath(connectorModule.getLocation());
196            }
197            return location;
198        */
199    }
```

Four spaces are used consistently for indention.

Issue 13: line 179 is 90 characters long, and the output could have been written on two lines.
Line 163 (declaration of the method) is 86 characters long, but it's not an issue because the names used cannot be shortened and it's better to have the throws statement on the same line of the declaration.

Issue 18: there is only one comment at line 170 not even correct in English. If there had been a Javadoc of the method, a single comment of better quality would have been sufficient to explain the code. So the programmer should add a Javadoc, correct the comment and maybe expand it a little bit to have a code adequately documented.

Issue 19: from line 185 to line 198 there is a commented out piece of code. There are no explanations included that specify the reason of the presence of this commented out code. It's only indicated presumably the version of the piece of code, maybe it's the third time that the programmer tries to implement that function, but this is not an useful information. The TODO tag probably means that this piece of code is yet to be finished or corrected, and this should be the reason why it's commented out, even if this is certainly not sufficient as an explanation. Moreover there is not a date when the code could be removed from the source file and until that part of the code remains commented out, if the moduleName string in input is null there should be a bug.

Issue 23: Javadoc doesn't cover the method under exam.

Issue 28: Methods called at line 166, 171 and 172 could have been declared as private instead of public, unless some other classes on Glassfish use them (we haven't done that check). A control of the call hierarchy of the method done only for ConnectorsUtil let us think that the methods at line 171 and 172 are only subprocedures of the getLocation method.
Instead the one at line 166 is also called by initializeSystemRars(), another method of ConnectorsUtil.

Issue 52: the method should raise a NullPointerException to deal with null parameters passed to the method. However, this part of method is currently going to be developed as can be seen in the commented-out section.

3.2.2 getAllSystemRAResourcesAndPools(Resources allResources)

```
325    /**
326     * Get all System RAR pools and resources
327     * @param allResources all configured resources
328     * @return Collection of system RAR pools
329     */
330    public static Collection<Resource> getAllSystemRAResourcesAndPools(Resources allResources) {
331        //Make sure that resources are added first and then pools.
332        List<Resource> resources = new ArrayList<Resource>();
333        List<Resource> pools = new ArrayList<Resource>();
334        for(Resource resource : allResources.getResources()){
335             if( resource instanceof ConnectorConnectionPool){
336                String raName = ((ConnectorConnectionPool)resource).getResourceAdapterName();
337                if( ConnectorsUtil.belongsToSystemRA(raName) ){
338                    pools.add(resource);
339                }
340            } else if( resource instanceof ConnectorResource){
341                String poolName = ((ConnectorResource)resource).getPoolName();
342                String raName = getResourceAdapterNameOfPool(poolName, allResources);
343                if( ConnectorsUtil.belongsToSystemRA(raName) ){
344                    resources.add(resource);
345                }
346            } else if (resource instanceof AdminObjectResource){ // jms-ra
347                String raName = ((AdminObjectResource)resource).getResAdapter();
348                if(ConnectorsUtil.belongsToSystemRA(raName)){
349                    resources.add(resource);
350                }
351            } //no need to list work-security-map as they are not deployable artifacts
352        }
353        resources.addAll(pools);
354        return resources;
355    }
```

Issue 8: At line 335, 5 spaces have been used and at line 336 6 spaces; this is not consistent with the rest of the method.

Issue 13: Line 330 (declaration of the method) is 97 characters long, but it is acceptable in order to express the purpose of the method.

Issue 18: There are some useful comments (the first and the third), instead the second one could be developed and rewritten as a sentence in English.

Issue 52: The method should raise a NullException when allResources is equal to null. Without raising this exception, in the worst case scenario, the method getResources() will be invoked on a null pointer causing an uncontrolled system failure.

3.2.3 isDynamicReconfigurationEnabled(ResourcePool pool)

```
408    public static boolean isDynamicReconfigurationEnabled(ResourcePool pool){
409        boolean enabled = false;
410        if(pool instanceof PropertyBag){
411            PropertyBag properties = (PropertyBag)pool;
412            Property property = properties.getProperty(ConnectorConstants.DYNAMIC_RECONFIGURATION_FLAG);
413            if(property != null){
414                try{
415                    if(Long.parseLong(property.getValue()) > 0){
416                        enabled = true;
417                    }
418                }catch(NumberFormatException nfe){
419                    _logger.log(Level.WARNING, "invalid.dynamic-reconfig.value", property.getValue());
420                }
421            }
422        }
423        return enabled;
424    }
```

Issue 13: line 412 is 92 characters long but it is more meaningful in this way even if this line is long more than 80 characters.

Issue 18/23: no comment are used to specify what the method is doing, neither a Javadoc is associated to the method. Even if the code it's relatively simple, there should have been some documentation.

3.2.4 getMergedActivationConfigProperties(EjbMessageBeanDescriptor msgDesc)

```
426     /**
427      * Prepares the name/value pairs for ActivationSpec. <p>
428      * Rule: <p>
429      * 1. The name/value pairs are the union of activation-config on
430      * standard DD (message-driven) and runtime DD (mdb-resource-adapter)
431      * 2. If there are duplicate property settings, the value in runtime
432      * activation-config will overwrite the one in the standard
433      * activation-config.
434      */
435     public static Set getMergedActivationConfigProperties(EjbMessageBeanDescriptor msgDesc) {
436
437         Set mergedProps = new HashSet();
438         Set runtimePropNames = new HashSet();
439
440         Set runtimeProps = msgDesc.getRuntimeActivationConfigProperties();
441         if (runtimeProps != null) {
442             Iterator iter = runtimeProps.iterator();
443             while (iter.hasNext()) {
444                 EnvironmentProperty entry = (EnvironmentProperty) iter.next();
445                 mergedProps.add(entry);
446                 String propName = (String) entry.getName();
447                 runtimePropNames.add(propName);
448             }
449         }
450
451         Set standardProps = msgDesc.getActivationConfigProperties();
452         if (standardProps != null) {
453             Iterator iter = standardProps.iterator();
454             while (iter.hasNext()) {
455                 EnvironmentProperty entry = (EnvironmentProperty) iter.next();
456                 String propName = (String) entry.getName();
457                 if (runtimePropNames.contains(propName))
458                     continue;
459                 mergedProps.add(entry);
460             }
461         }
462
463         return mergedProps;
464     }
```

Issue 11: the if statement at line 457, having only one statement inside it, doesn't have curly braces. While this is not a bug of the code, it's preferred to write the curly braces even in this situation.

Issue 33: declarations don't always appear at beginning of blocks (see line 446 and line 451).

The line containing the declaration of the method is longer than 80 characters but there's no way to fix this problem, so it's not an issue (it's less than 120 characters).

The while statements adopt the iterator abstraction and they are well defined and free of bugs.
The code is free of any implicit type conversions and cast is applied correctly at line 444, 455, 456.
While it's not an issue contained in the checklist, the use of the "continue" statement at line 458 is usually not recommended. However its use in the examined method in this case is correct and the code still remains easily understandable.

Issue 52: it should be handled the case in which the parameter passed to the method is null with a try/catch block and a null exception, to prevent errors at line 440 and 451.

3.2.5 getShutdownTimeout(ConnectorService connectorService)

```
478      * Gets the shutdown-timeout attribute from domain.xml
479      * via the connector server config bean.
480      * @param connectorService connector-service configuration
481      * @return long shutdown timeout (in mill-seconds)
482      */
483     public static long getShutdownTimeout(ConnectorService connectorService)  {
484         int shutdownTimeout;
485 
486         try {
487             if (connectorService == null) {
488                 //Connector service element is not specified in
489                 //domain.xml and hence going with the default time-out
490                 shutdownTimeout =
491                         ConnectorConstants.DEFAULT_RESOURCE_ADAPTER_SHUTDOWN_TIMEOUT;
492                 if(_logger.isLoggable(Level.FINE)) {
493                     _logger.log(Level.FINE, "Shutdown timeout set to "+  shutdownTimeout + " through default");
494                 }
495             } else {
496                 shutdownTimeout = Integer.parseInt(connectorService.getShutdownTimeoutInSeconds());
497                 if(_logger.isLoggable(Level.FINE)) {
498                     _logger.log(Level.FINE, "Shutdown timeout set to " + shutdownTimeout + " from domain.xml");
499                 }
500             }
501         } catch (Exception e) {
502             _logger.log(Level.WARNING, "error_reading_connectorservice_elt", e);
503             //Going ahead with the default timeout value
504             shutdownTimeout = ConnectorConstants.DEFAULT_RESOURCE_ADAPTER_SHUTDOWN_TIMEOUT;
505         }
506         return shutdownTimeout * 1000L;
507     }
```

Issue13: line 493 and line 498 are 91 characters long, both should be better with a line break after comma, having so first line 23 characters long and the second of 68 characters.

Issue 15: line 490, the break should have been made before the operator and not after . Anyway the indentation of line 491 is correct (8 spaces).
Comments adequately explain what this piece of code is doing because all the method used are covered by the java documentation. The Javadoc is present unlike in many other methods of ConnectorsUtil and provides useful informations.
The cast from int to long is explicit and it's done with the use of capital L after the number to be converted, at line 506.

3.2.6 getTransactionIsolationInt(int tranIsolation)

```
537    public static String getTransactionIsolationInt(int tranIsolation) {
538
539        if(tranIsolation == Connection.TRANSACTION_READ_UNCOMMITTED){
540            return "read-uncommited";
541        } else if(tranIsolation == Connection.TRANSACTION_READ_COMMITTED){
542            return "read-committed";
543        } else if(tranIsolation == Connection.TRANSACTION_REPEATABLE_READ){
544            return "repeatable-read";
545        } else if(tranIsolation == Connection.TRANSACTION_SERIALIZABLE){
546            return "serializable";
547        } else {
548            throw new RuntimeException("Invalid transaction isolation; the transaction "
549                    + "isolation level can be empty or any of the following: "
550                    + "read-uncommitted, read-committed, repeatable-read, serializable");
551        }
552    }
553
```

Kernington and Ritchie brace style is adopted with else if statements written on the same line of the } of the previous else if.

Issue 13: line 548 is 89 characters long and it's not a situation where it's impossible to avoid the problem and so we fix the maximum line lenght at 120 characters. By simply moving the word "transactions" of the output message on the next line this little problem would be fixed.

Issue 15: line 548, 549 and 550 don't break after an operator but before the string concatenation operator.

Issue 23: this method doesn't have an associated javadoc.
The outputs doesn't contain any spelling or grammatical errors or line stepping or spacing errors, and the meaning of the error message is clear.

The method uses else if statement many times, but this is not one of the situations described as brutish programming and there's not a way to optimize the code without losing the compactness. If there were more cases than four, a switch statement could have improved the code, but however it's not an issue.

3.2.7 getReservePrefixedJNDINameForResource(String compId , String resourceName , JavaEEResourceType resType)

```
561  public static String getReservePrefixedJNDINameForResource(String compId, String resourceName, JavaEEResourceType resType) {
562      String prefix = null;
563      String prefixPart1 = null;
564      String prefixPart2 = null;
565
566      if(resType!=null) {
567          switch (resType) {
568              case DSD :
569                  prefixPart1 = ConnectorConstants.RESOURCE_JNDINAME_PREFIX;
570                  prefixPart2 = ConnectorConstants.DATASOURCE_DEFINITION_JNDINAME_PREFIX;
571              break;
572              case MSD :
573                  prefixPart1 = ConnectorConstants.RESOURCE_JNDINAME_PREFIX;
574                  prefixPart2 = ConnectorConstants.MAILSESSION_DEFINITION_JNDINAME_PREFIX;
575                  break;
576              case CFD :
577                  prefixPart1 = ConnectorConstants.RESOURCE_JNDINAME_PREFIX;
578                  prefixPart2 = ConnectorConstants.CONNECTION_FACTORY_DEFINITION_JNDINAME_PREFIX;
579                  break;
580              case DSDPOOL:
581                  prefixPart1 = ConnectorConstants.POOLS_JNDINAME_PREFIX;
582                  prefixPart2 = ConnectorConstants.DATASOURCE_DEFINITION_JNDINAME_PREFIX;
583                  break;
584              case CFDPOOL:
585                  prefixPart1 = ConnectorConstants.POOLS_JNDINAME_PREFIX;
586                  prefixPart2 = ConnectorConstants.CONNECTION_FACTORY_DEFINITION_JNDINAME_PREFIX;
587                  break;
588              case JMSCFDD:
589                  prefixPart1 = ConnectorConstants.RESOURCE_JNDINAME_PREFIX;
590                  prefixPart2 = ConnectorConstants.JMS_CONNECTION_FACTORY_DEFINITION_JNDINAME_PREFIX;
591                  break;
592              case JMSCFDDPOOL:
593                  prefixPart1 = ConnectorConstants.POOLS_JNDINAME_PREFIX;
594                  prefixPart2 = ConnectorConstants.JMS_CONNECTION_FACTORY_DEFINITION_JNDINAME_PREFIX;
595                  break;

596                 case JMSDD:
597                     prefixPart1 = ConnectorConstants.RESOURCE_JNDINAME_PREFIX;
598                     prefixPart2 = ConnectorConstants.JMS_DESTINATION_DEFINITION_JNDINAME_PREFIX;
599                     break;
600                 case AODD:
601                     prefixPart1 = ConnectorConstants.RESOURCE_JNDINAME_PREFIX;
602                     prefixPart2 = ConnectorConstants.ADMINISTERED_OBJECT_DEFINITION_JNDINAME_PREFIX;
603                     break;
604             }
605      }
606
607      if(compId == null || compId.equals("")){
608          prefix =  prefixPart1 + prefixPart2;
609      }else{
610          prefix = prefixPart1 + prefixPart2 + compId +"/";
611      }
612      return getReservePrefixedJNDIName(prefix, resourceName);
613  }
```

Issue 8: the break statement at line 571 is indented in a wrong way. There is also an extra space at line 566,567 between if and the parenthesis of its condition.

Issue 18: There is no comment for this class, none of its instructions is explained. It should have been useful to know what is the string “compId”.

This method uses the switch statement without brute forcing, it seems there's no better way to write this function even if it's easy to make mistakes by duplicating the code.

Issue 52: The method should raise some NullExceptions, instead deals with exceptional cases as normal ones.
If an input parameter is null there should always be an exception signalizing the situation. In this block of code instead some parameters are controlled without exception just before using them. No worst case scenario is considered, for instance at line 566 there is a control to check whether the input parameter resType is Null but the case in which resType is equal to null isn't dealt and this situation will surely create a problem.

Issue 55: This switch statement hasn't a default branch so if resType doesn't match to anything of the listened values the method won't behave as expected.

4. Any other problem you have highlighted

The entire method getAllSystemRAResourcesAndPools(Resources allResources) at line 330 in the ConnectorsUtil class is built on top of a bad approach to resolve the problem it is facing.
There are a lot of constructs like:

```
if( object instanceof A certain class)
    cast the object to that class
```

This method represents a mistake caused by a bad software design. This situation becomes also worse because it is repeated for 3 times in a for loop in the intention of 'enumerating' all possible cases.
An ideal solution would be to change the design of the classes so that there can either be a common base class or a generic method.

The method getReservePrefixedJNDINameForResource(String compId , String resourceName , JavaEEResourceType resType) at its very last statements invokes the following private class instruction:

```
622    private static String getReservePrefixedJNDIName(String prefix, String resourceName) {
623        return prefix + resourceName;
624    }
```

This is redundant and useless. At first place it isn't a getter and its name is deviating. This method is performing a simple concatenation between two strings and should be erased. Its presence increase the number of the class methods that is already high and doesn't add anything from a semantic point of view.