Code Inspection

Matteo Maria Fusi, Matteo Locatelli

0.1 Document version

- 1.0 - 5/1/2016: first release

0.2 Time Spent

- Matteo M. Fusi: ~10h
- Matteo Locatelli: ~10h

Contents

1 Classes assigned

Four methods have been assigned to us, all belonging to the BaseContainer class, located at *appserver/ejb/ejb-container/src/main/java/com/sun/ejb/containers/BaseContainer.java.* The methods that we analyzed are the following:

- **Name:** createEjbInstanceAndContext( )

  Start line: 1682

  End line: 1729

- **Name:** injectEjbInstance(EJBContextImpl context)

  Start line: 1740

  End line: 1778

- **Name:** getTargetObject(byte [ ] instanceKey , String generatedRemote-BusinessIntf)

  Start line: 1794

  End line: 1826

- **Name:** preInvoke(EjbInvocation inv)

  Start line: 1922

  End line: 2009

2 Functional role of the assigned set of classes

2.1 createEjbInstanceAndContext

This method, as its name explains, has the task to create an EJB (Enterprise Java Bean) instance and its context. When an EJB is needed the *invocation-Manager* calls the *preInvoke* method, that manages all the tasks that have to be done before the EJB is actually created *(invocationManager.preInvoke(ejbInv);* statement at line 1693). After that, the method checks if the java beans needs another bean injected (the *if* statement at line 1695), in this case an EJB with injection is created (the *then* block at lines 1696-1697), otherwise a "normal" java bean is created without any beans injected into it (the *else* block at lines 1699-1701). If the invocation has not been done correctly an exception is generated and it is caught by the catch block at lines 1705-1706 and an *InvocationTargetEx-ception* is thrown. Then, if the invocation has been done correctly (if statement at line 1709) the method *postInvoke* is called (line 1711) and it manages all the tasks that have to be done after the invocation of the EJB. If the calling of the *postInvoke* method has not been done right, an exception is thrown and caught by the *catch* block at lines 1713-1719. Inside this *catch* block, if the variable *success* has value true (meaning that the invocation of the EJB was correct) an *InvocationTargetException* is thrown (line 1715), otherwise a log is shown (line

1717) to evidence the presence of an error of the *postInovke* calling. After that, the context of the EJB is created using the method *constructEJBContextImpl* (line 1723) and if the java bean has another bean injected (*if* statement at line 1724) the injection context is created using the method *setJCDIInjectionContext* (line 1725). Finally the context of the EJB is returned (line 1728).

2.2 injectEjbInstance

This method has the task to inject an instance of an EJB inside another bean. Firstly, the method checks if the bean that is been injected has another bean inside itself (*if* statement at line 1746), in this case this method is also called for the bean that is injected (this method is called recursively for each bean injected, statement at line 1748). Then interceptor instances are created corrisponding to the classes that can be intercepted by the bean when a request is made (lines 1750 - 1761). Otherwise, if the condition of the *if* statement at line 1746 is false, the method checks if the context contains an EJB (*if* statement at line 1764), if so a new instance is injected (statement at line 1765). Then interceptor instances are created and injected (statements at lines 1768 - 1773). Finally, the interceptor instances of the context are inizialized.

2.3 getTargetObject

This method returns an EJB when a remote invocation arrives. Firstly, the method checks if the invocation is local or remote (line 1798). If it's local (*if* statement at line 1809) the implementation of the EJB is returned, otherwise the remote EJB implementation is returned.

2.4 preInvoke

This method firstly checks if the container is ready to accept the request from a client (line 1929 - 1934). Then if the invocation is missing some information, this method tries to provide it (or throws an exception if it is impossible, lines 1942 - 1949). After that, the container is associated to the invocation (line 1952) and if the invocation has to be done synchronously, the method checks if the user has the privileges to access the information he requested (lines 1958 - 1964). Then the invocation attributes are set (lines 1969 - 1977). If one of the previous operations generates an exception, it is caught, the error is shown using a logger and a new unchecked exception is thrown (lines 1996 - 2008).

3 List of issues found

In this section the issues found in the method that were assigned to us are reported, following the checklist in the assignment track. Firstly there is an analysis concerning the hole class, then the issues of every method are described.

3.1 Class general analysis

Java Source Files

20. Each Java source file contains a single public class or interface.

    - This condition is not respected because there are three public classes inside the analyzed source files: the *BaseContainer* class defined at line 182, the *PreInvokeException* class defined at line 4998 and the *ContainerInfo* class defined at line 5010.

21. The public class is the first class or interface in the file.

    - This condition is respected because the first class in the file is a public class, but, as said in the previous point, there are multiple public classes inside this file.

22. Check that the external program interfaces are implemented consistently with what is described in the javadoc.

    - This condition is respected beacuse every method of the used external interfaces is implemented correctly.

23. Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

    - This condition is not respected, because a lot of methods implemented in this file are missing the javadoc.

Package and Import Statements

24. If any package statements are needed, they should be the first non-comment statements. Import statements follow.

    - This condition is respected.

Class and Interface Declarations

25. The class or interface declarations shall be in the following order:

    A. class/interface documentation comment
    B. class or interface statement
    C. class/interface implementation comment, if necessary
    D. class (static) variables
        a. first public class variables
        b. next protected class variables
        c. next package level (no access modifier)

        d. last private class variables

    E. instance variables

        a. first public instance variables
        e. next protected instance variables
        f. next package level (no access modifier)
        g. last private instance variables

    F. constructors

    G. methods

- There are eight classes in this file:
  - *BaseContainer*: point C is not respected because there isn't an implementation comment. Points D and E are not respected because class variables are not defined in that order. Other points are respected.
  - *JndiInfo*: points A and C are not respected because there aren't comments explaining the functionalities of this class. Point from D to G are not respected because the constructor and the methods are declared before the variables.
  - *PreInvokeException*: point C is not respected because there isn't an implementation comment.
  - *ContainerInfo*: point C is not respected because there isn't an implementation comment.
  - *BeanContext*: points A and C are not respected.
  - *CallFlowInfoImpl*: points A and C are not respected.
  - *RemoteBusinessIntfInfo*: points A and C are not respected.
  - *SafeProperties*: points A and C are not respected.

26. Methods are grouped by functionality rather than by scope or accessibility.

    - This condition is respected.

27. Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

    - The main issue about this point is that the classes and the mothods are too long, for example the BaseContainer class is almost 5000 lines long and its constructor is almost 300 lines long.

3.2 createEjbInstanceAndContext

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.

- throws Exception (line 1682): *Exception* is not a meaningful name, the exception should have a name that represents what type of exception the method throws.
- Object instance (line 1686): this Object variable should have a name that explains its functionalities, *instance* is a too much generic word.
- boolean success (line 1690): *success* is too generic, it would be useful to know what success is about.
- Throwable th (line 1705): *th* isn't a meaningful name, the throwable object should have a name that represents the functionality of the object.
- Throwable t (line 1713): *t* isn't a meaningful name, the throwable object should have a name that represents the functionality of the object.

2. If one character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.

   - Throwable t (line 1713): *t* is a name for a Throwable variable that is only used twice, but should have a meaningful name as explained in the point above.

3. Class names are nouns, in mixed case, with the first letter of each word in capitalized.

   - In this method there isn't any error of this kind.

4. Interface names should be capitalized like classes.

   - In this method there isn't any error of this kind.

5. Method names should be verbs, with the first letter of each addition word capitalized.

   - In this method there isn't any error of this kind.

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.

   - In this method there isn't any error of this kind.

7. Constants are declared using all uppercase with words separated by an underscore.

   - In this method there isn't any error of this kind.

Indention

8. Three or four spaces are used for indentation and done so consistently.

    - In this method there isn't any error of this kind, because always four spaces are used for indentation.

9. No tabs are used to indent.

    - In this method there isn't any error of this kind, because only spaces are used for indentation.

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).

    - In this method there isn't any error of this kind, because the "Kernighan and Ritchie" style is always used.

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

    - In this method there isn't any error of this kind.

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).

    - In this method, there shouldn't be the blank line at lines 1684 and 1687, but there should be one at line 1690 separating the variables declaration section of the method from the following section. Other blank lines are used correctly. Optional comments are not used.

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

    - In this method, each line is shorter than 80 characters.

14. When line length must exceed 80 characters, it does NOT exceed 120 characters.

    - There isn't any error of this kind because the previous point is respected.

Wrapping Lines

15. Line break occurs after a comma or an operator.

    - This condition is respected in this method.

16. Higher-level breaks are used.

    - This condition is respected in this method.

17. A new statement is aligned with the beginning of the expression at the same level as the previous line.

    - This condition is respected in this method.

Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

    - This condition is not respected at all, because this method lacks comments useful to understand the general functionality of the method and what blacks of code (that aren't easy to understand) 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

    - This condition is respected because there isn't commented out code in this method.

Inizialization and declarations

28. Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).

    - This condition is respected in this method, because all the variables declared are of the correct type.

29. Check that variables are declared in the proper scope.

    - The *ejbBundle* variable declared at line 1683 should be declared inside the scope of the try block that starts at line 1691, because the *ejbBundle* variable is used only inside this try block.
    - The *ctx* variable declared at line 1688 should be declared inside the scope of the try block that starts at line 1691, because the *ctx* variable is used only inside this try block.

30. Check that constructors are called when a new object is desired.

- This condition is respected in this method.

31. Check that all object references are initialized before use.
    - This condition is respected in this method.
32. Variables are initialized where they are declared, unless dependent upon a computation.
    - This condition is respected in this method.
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.
    - The variable *contextImpl* isn't declared at the beginning of the block, so it should be declared at the begginning of the method where other variables such as *ejbBundle* and *jcdiCtx* are declared.

Method calls

34. Check that parameters are presented in the correct order.
    - This condition is respected in this method because all the methods are called correctly.
35. Check that the correct method is being called, or should it be a different method with a similar name.
    - The method *setJCDIInjectionContext(jcdiCtx);* at line 1725 doesn't seem to exist in the javadoc of the used version of Glassfish.
36. Check that method returned values are used properly.
    - This condition is respected in this method.

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).
    - This condition is respected in this method because no array is used.
38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds.
    - This condition is respected in this method because no array is used.
39. Check that constructors are called when a new array item is desired.
    - This condition is respected in this method because no array is used.

Object Comparison

40. Check that all objects (including Strings) are compared with "equals" and not with "==".

    - This condition is respected in this method because the "==" operator isn't used.

Output format

41. Check that displayed output is free of spelling and grammatical errors.

    - This condition is respected in this method because the only output is "" produced by the logger at line 1717

42. Check that error messages are comprehensive and provide guidance as to how to correct the problem.

    - This condition is not respected, because the logger *_logger.log(Level.WARNING, "", t)* at line 1717 should explain what has generated a warning, so instead of the "" output there should be a message describing what type of error occured.

43. Check that the output is formatted correctly in terms of line stepping and spacing.

    - This condition is respected in this method because the only output is "" produced by the logger at line 1717

Computation, Comparisons and Assignments

44. Check that the implementation avoids "brutish programming":

    - This condition is respected in this method because the method doesn't contain any "brutish" block of code.

45. Check order of computation/evaluation, operator precedence and parenthesizing.

    - This condition is respected in this method.

46. Check the liberal use of parenthesis is used to avoid operator precedence problems.

    - in the istruction *if((jcdiService != null) && jcdiService.isJCDIEnabled(ejbBundle))* at line 1695 the paranthesis sorrounding the condition *jcdiService != null* aren't necessary.

47. Check that all denominators of a division are prevented from being zero.

- This condition is respected in this method because there aren't divisions.

48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.

    - This condition is respected in this method because there aren't arithmetic operations

49. Check that the comparison and Boolean operators are correct.

    - This condition is respected in this method.

50. Check throw-catch expressions, and check that the error condition is actually legitimate.

    - This condition is respected in this method.

51. Check that the code is free of any implicit type conversions.

    - The instruction *EjbBundleDescriptorImpl ejbBundle = ejbDescriptor.getEjbBundleDescriptor();* at line 1683 contains an implicit type conversion from type *EjbBundleDescriptor* to type *EjbBundleDescriptorImpl.*

Exceptions

52. Check that the relevant exceptions are caught.

    - This condition is respected in this method.

53. Check that the appropriate action are taken for each catch block.

    - This condition is respected in this method because every catch block is corrected.

Flow of Control

54. In a switch statement, check that all cases are addressed by break or return.

    - This condition is respected in this method because there is no switch statement.

55. Check that all switch statements have a default branch.

    - This condition is respected in this method because there is no switch statement.

56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

    - This condition is respected in this method because there are no loops.

Files

57. Check that all files are properly declared and opened.
    - This condition is respected in this method because no file is used.
58. Check that all files are closed properly, even in the case of an error.
    - This condition is respected in this method because no file is used.
59. Check that EOF conditions are detected and handled correctly.
    - This condition is respected in this method because no file is used.
60. Check that all file exceptions are caught and dealt with accordingly.
    - This condition is respected in this method because no file is used.

3.3 injectEjbInstance

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.
    - throws Exception (line 1740): *Exception* is not a meaningful name, the exception should have a name that represents what type of exception the method throws.
2. If one character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.
    - In this method there isn't any error of this kind.
3. Class names are nouns, in mixed case, with the first letter of each word in capitalized.
    - In this method there isn't any error of this kind.
4. Interface names should be capitalized like classes.
    - In this method there isn't any error of this kind.
5. Method names should be verbs, with the first letter of each addition word capitalized.
    - In this method there isn't any error of this kind.

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.

   - In this method there isn't any error of this kind.

7. Constants are declared using all uppercase with words separated by an underscore.

   - In this method there isn't any error of this kind, because constants aren't used.

Indention

8. Three or four spaces are used for indentation and done so consistently.

   - In this method this condition is not respected, because the indentation is not correct, for example at line 1742 the instruction should be indented with 4 spaces but it is indented with 8 spaces and at line 1758 too much spaces are used. Furthermore, at lines 1748, 1771, 1772 and 1773 tabs are used to indent instructions.

9. No tabs are used to indent.

   - In this method this condition is not respected because at lines 1748, 1771, 1772 and 1773 tabs are used to indent instructions.

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).

    - In this method there isn't any error of this kind, because the "Kernighan and Ritchie" style is always used.

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

    - In this method there isn't any error of this kind.

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).

- In this method this condition is not respected because a blank line is used to separate almost every instruction from the next one. All these blank lines should be removed, the only correct one is the one at line 1745 that is used to separate the declaration block from the rest of the code. Optional comments are not used.

13. Where practical, line length does not exceed 80 characters.
    - In this method, each line is shorter than 80 characters.
14. When line length must exceed 80 characters, it does NOT exceed 120 characters.
    - There isn't any error of this kind because the previous point is respected.

Wrapping Lines

15. Line break occurs after a comma or an operator.
    - This condition is respected in this method.
16. Higher-level breaks are used.
    - This condition is respected in this method.
17. A new statement is aligned with the beginning of the expression at the same level as the previous line.
    - This condition is respected in this method.

Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.
    - This condition is not respected at all, because this method lacks comments useful to understand the general functionality of the method and what blacks of code (that aren't easy to understand) are doing. The only comments at lines 1755 and 1756 are not helpful to understand what the next block of code does.
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
    - This condition is respected because there isn't commented out code in this method.

Inizialization and declarations

28. Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).
    - This condition is respected in this method, because all the variables declared are of the correct type.
29. Check that variables are declared in the proper scope.
    - This condition is respected in this method, because all the variables are declared in the proper scope.
30. Check that constructors are called when a new object is desired.
    - This condition is respected in this method.
31. Check that all object references are initialized before use.
    - This condition is respected in this method.
32. Variables are initialized where they are declared, unless dependent upon a computation.
    - This condition is respected in this method.
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.
    - The declaration of the *interceptorClasses* variable at line 1750 should be moved to the beginning of the block at line 1747.

Method calls

34. Check that parameters are presented in the correct order.
    - This condition is respected in this method because all the methods are called correctly.
35. Check that the correct method is being called, or should it be a different method with a similar name.
    - The method *setInterceptorInstances* at line 1776 doesn't seem to exist in the javadoc of the used version of Glassfish.
36. Check that method returned values are used properly.
    - This condition is respected in this method.

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).
    - This condition is respected in this method.
38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds.
    - This condition is respected in this method.
39. Check that constructors are called when a new array item is desired.
    - This condition is respected in this method.

Object Comparison

40. Check that all objects (including Strings) are compared with "equals" and not with "==".
    - This condition is respected in this method because the "==" operator isn't used.

Output format

41. Check that displayed output is free of spelling and grammatical errors.
    - This condition is respected in this method because it doesn't have any displayed output.
42. Check that error messages are comprehensive and provide guidance as to how to correct the problem.
    - This condition is respected in this method because it doesn't have any displayed output.
43. Check that the output is formatted correctly in terms of line stepping and spacing.
    - This condition is respected in this method because it doesn't have any displayed output.

Computation, Comparisons and Assignments

44. Check that the implementation avoids "brutish programming":
    - This condition is respected in this method because the method doesn't contain any "brutish" block of code.
45. Check order of computation/evaluation, operator precedence and parenthesizing.
    - This condition is respected in this method.
46. Check the liberal use of parenthesis is used to avoid operator precedence problems.
    - in the istruction *if((jcdiService != null) && jcdiService.isJCDIEnabled(ejbBundle))* at line 1746 the paranthesis sorrounding the condition *jcdiService != null* aren't necessary.
47. Check that all denominators of a division are prevented from being zero.
    - This condition is respected in this method because there aren't divisions.
48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.
    - This condition is respected in this method because there aren't arithmetic operations.
49. Check that the comparison and Boolean operators are correct.
    - This condition is respected in this method.
50. Check throw-catch expressions, and check that the error condition is actually legitimate.
    - This condition is respected in this method because there isn't any throw-catch expression.
51. Check that the code is free of any implicit type conversions.
    - The instruction *EjbBundleDescriptorImpl ejbBundle = ejbDescriptor.getEjbBundleDescriptor();* at line 1742 contains an implicit type conversion from type *EjbBundleDescriptor* to type *EjbBundleDescriptorImpl*.
    - The instruction *interceptorInstances[i] = jcdiService.createInterceptorInstance(interceptorClasse ejbBundle);* at lines 1757 and 1758 contains an implicit type conversion from type *EjbBundleDescriptor* to type *BundleDescriptor* of the variable *ejbBundle*.

- The instruction *injectionManager.injectInstance(context.getEJB(), ejb-Descriptor, false);* at line 1765 contains an implicit type conversion from type *EjbDescriptor* to type *JndiNameEnvironment* of the variable *ejbDescriptor.*
- The instruction *injectionManager.injectInstance(interceptorInstance, ejbDescriptor, false);* at lines 1771 and 1772 contains an implicit type conversion from type *EjbDescriptor* to type *JndiNameEnvironment* of the variable *ejbDescriptor.*

Exceptions

52. Check that the relevant exceptions are caught.
    - This condition is respected in this method.
53. Check that the appropriate action are taken for each catch block.
    - This condition is respected in this method because there isn't any catch block.

Flow of Control

54. In a switch statement, check that all cases are addressed by break or return.
    - This condition is respected in this method because there is no switch statement.
55. Check that all switch statements have a default branch.
    - This condition is respected in this method because there is no switch statement.
56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.
    - This condition is respected in this method.

Files

57. Check that all files are properly declared and opened.
    - This condition is respected in this method because no file is used.
58. Check that all files are closed properly, even in the case of an error.
    - This condition is respected in this method because no file is used.
59. Check that EOF conditions are detected and handled correctly.

- This condition is respected in this method because no file is used.

60. Check that all file exceptions are caught and dealt with accordingly.

    - This condition is respected in this method because no file is used.

3.4 getTargetObject

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.

    - In this method there isn't any error of this kind.

2. If one character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.

    - In this method there isn't any error of this kind.

3. Class names are nouns, in mixed case, with the first letter of each word in capitalized.

    - In this method there isn't any error of this kind.

4. Interface names should be capitalized like classes.

    - In this method there isn't any error of this kind.

5. Method names should be verbs, with the first letter of each addition word capitalized.

    - In this method there isn't any error of this kind.

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.

    - In this method there isn't any error of this kind.

7. Constants are declared using all uppercase with words separated by an underscore.

    - In this method there isn't any error of this kind.

Indention

8. Three or four spaces are used for indentation and done so consistently.
    - In this method there isn't any error of this kind, because always four spaces are used for indentation.
9. No tabs are used to indent.
    - In this method there isn't any error of this kind, because only spaces are used for indentation.

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).
    - In this method there isn't any error of this kind, because the "Kernighan and Ritchie" style is always used.
11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.
    - In this method there isn't any error of this kind.

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).
    - In this method, there shouldn't be the blank line at lines 1804, 1807, 1808 and 1823, but there should be one at line 1798 separating the variables declaration section of the method from the following section. Optional comments are not used.
13. Where practical, line length does not exceed 80 characters.
    - In this method, each line is shorter than 80 characters.
14. When line length must exceed 80 characters, it does NOT exceed 120 characters.
    - There isn't any error of this kind because the previous point is respected.

Wrapping Lines

15. Line break occurs after a comma or an operator.

    - The line break at line 1819 should not be used. If the instructions at lines 1819 and 1820 were written on a single line the length of the line would be less than 80 characters so it would be right.

16. Higher-level breaks are used.

    - This condition is respected in this method.

17. A new statement is aligned with the beginning of the expression at the same level as the previous line.

    - This condition is respected in this method.

Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

    - This condition is not completely respected, because this method lacks comments useful to understand what some blacks of code are doing. The only two comments used are at lines 1811 and 1817; they help to understand the next statements but other lines of code are unexplained.

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

    - This condition is respected because there isn't commented out code in this method.

Inizialization and declarations

28. Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).

    - This condition is respected in this method, because all the variables declared are of the correct type.

29. Check that variables are declared in the proper scope.

    - This condition is respected in this method.

30. Check that constructors are called when a new object is desired.

- This condition is respected in this method.

31. Check that all object references are initialized before use.
    - This condition is respected in this method.
32. Variables are initialized where they are declared, unless dependent upon a computation.
    - This condition is respected in this method.
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.
    - The variable *remoteHomeView* isn't declared at the beginning of the block, so it should be declared at the begginning of the method before the *externalPreInvoke();* statement.

Method calls

34. Check that parameters are presented in the correct order.
    - This condition is respected in this method because all the methods are called correctly.
35. Check that the correct method is being called, or should it be a different method with a similar name.
    - The condition is respected.
36. Check that method returned values are used properly.
    - This condition is respected in this method.

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).
    - This condition is respected in this method.
38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds.
    - This condition is respected in this method.
39. Check that constructors are called when a new array item is desired.
    - This condition is respected in this method.

Object Comparison

40. Check that all objects (including Strings) are compared with “equals” and not with “==”.

    - This condition is not respected in this method because the “==” operator is used at lines 1798 (*boolean remoteHomeView = (generatedRemoteBusinessIntf == null);* statement) and 1799 (*if (instanceKey.length == 1 && instanceKey[0] == HOME_KEY)* statement).

Output format

41. Check that displayed output is free of spelling and grammatical errors.

    - This condition is respected because this method doesn’t produce any output.

42. Check that error messages are comprehensive and provide guidance as to how to correct the problem.

    - This condition is respected because this method doesn’t produce any output.

43. Check that the output is formatted correctly in terms of line stepping and spacing.

    - This condition is respected because this method doesn’t produce any output.

Computation, Comparisons and Assignments

44. Check that the implementation avoids “brutish programming”:

    - This condition is respected in this method because the method doesn’t contain any “brutish” block of code.

45. Check order of computation/evaluation, operator precedence and parenthesizing.

    - This condition is respected in this method.

46. Check the liberal use of parenthesis is used to avoid operator precedence problems.

    - This condition is not respected in this method because the statement *boolean remoteHomeView = (generatedRemoteBusinessIntf == null);* at line 1798 doesn’t need the parathesis.

47. Check that all denominators of a division are prevented from being zero.

    - This condition is respected in this method because there aren't divisions.

48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.

    - This condition is respected in this method because there aren't arithmetic operations

49. Check that the comparison and Boolean operators are correct.

    - This condition is respected in this method.

50. Check throw-catch expressions, and check that the error condition is actually legitimate.

    - This condition is respected in this method because there isn't any throw-catch expression.

51. Check that the code is free of any implicit type conversions.

    - The instructions *ejbHomeImpl.getEJBHome()* at line 1801 and *ejbRemoteBusinessHomeImpl.getEJBHome()* at line 1802 contain an implicit type conversion from type *EJBHomeImpl* (the type returned by the method *getEJBHome()*) to type *Remote* (the type returned by the currently inspected method).

Exceptions

52. Check that the relevant exceptions are caught.

    - This condition is not respected in this method because the *NoSuchObjectLocalException* exception specified in the JavaDoc of the mothod isn't neither caught nor thrown.

53. Check that the appropriate action are taken for each catch block.

    - This condition is respected in this method because there isn't any catch block.

Flow of Control

54. In a switch statement, check that all cases are addressed by break or return.
    - This condition is respected in this method because there is no switch statement.
55. Check that all switch statements have a default branch.
    - This condition is respected in this method because there is no switch statement.
56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.
    - This condition is respected in this method because there are no loops.

Files

57. Check that all files are properly declared and opened.
    - This condition is respected in this method because no file is used.
58. Check that all files are closed properly, even in the case of an error.
    - This condition is respected in this method because no file is used.
59. Check that EOF conditions are detected and handled correctly.
    - This condition is respected in this method because no file is used.
60. Check that all file exceptions are caught and dealt with accordingly.
    - This condition is respected in this method because no file is used.

3.5 preInvoke

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.
    - In this method there isn't any error of this kind.
2. If one character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.
    - In this method there isn't any error of this kind.

3. Class names are nouns, in mixed case, with the first letter of each word in capitalized.
    - In this method there isn't any error of this kind.
4. Interface names should be capitalized like classes.
    - In this method there isn't any error of this kind.
5. Method names should be verbs, with the first letter of each addition word capitalized.
    - In this method there isn't any error of this kind.
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.
    - In this method there isn't any error of this kind.
7. Constants are declared using all uppercase with words separated by an underscore.
    - In this method there isn't any error of this kind.

Indention

8. Three or four spaces are used for indentation and done so consistently.
    - In this method there isn't any error of this kind, because always four spaces are used for indentation.
9. No tabs are used to indent.
    - In this method there isn't any error of this kind, because only spaces are used for indentation.

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).
    - In this method there isn't any error of this kind, because the "Kernighan and Ritchie" style is always used.
11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.
    - In this method there isn't any error of this kind.

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).

    - Blank lines are used in a correct way.
    - Instead of a blank line, a short comment could have been a better solution.

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

    - Line 1947: better make a new line with the `inv.method` parameter

14. When line length must exceed 80 characters, it does NOT exceed 120 characters.

    - Some lines exceeed 80 characters, but never 120.

Wrapping Lines

15. Line break occurs after a comma or an operator.

    - The condition is respected is respected in this method.

16. Higher-level breaks are used.

    - This condition is respected in this method.

17. A new statement is aligned with the beginning of the expression at the same level as the previous line.

    - This condition is respected in this method.

Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

    - There are some comments, but they don't explain well what the code does and why.

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

    - At line 1978 there's a line of code commented out but it's not explained why and the date of the comment is missing.

Other issues

- At line 1988 there's a mistyping in the comment: `preInovkeTxStatus` should be `preInvokeTxStatus`.

Inizialization and declarations

28. Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).
    - This condition is respected in this method, because all the variables declared are of the correct type.
29. Check that variables are declared in the proper scope.
    - This condition is respected in this method.
30. Check that constructors are called when a new object is desired.
    - This condition is respected in this method.
31. Check that all object references are initialized before use.
    - This condition is respected in this method.
32. Variables are initialized where they are declared, unless dependent upon a computation.
    - This condition is respected in this method.
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.
    - Declarations are done after a serie of exception checking for the sake of saving memory. In this way memory is not allocated if it will not be used.

Method calls

34. Check that parameters are presented in the correct order.
    - This condition is respected in this method because all the methods are called correctly.
35. Check that the correct method is being called, or should it be a different method with a similar name.
    - This condition is respected.
36. Check that method returned values are used properly.
    - This condition is respected in this method.

Arrays

In the `preInvoke` method arrays don't exists.

Object Comparison

40. Check that all objects (including Strings) are compared with "equals" and not with "==".

    - At line 1929 there's a comparison `containerState != CONTAINER_STARTED`, which is illegal.

Output format

41. Check that displayed output is free of spelling and grammatical errors.

    - Log messages are correct.

42. Check that error messages are comprehensive and provide guidance as to how to correct the problem.

    - Log messages just show the caught exception. They don't provide any kind of solution, so this condition is not respected.

43. Check that the output is formatted correctly in terms of line stepping and spacing.

    - The output is well formatted.

Computation, Comparisons and Assignments

44. Check that the implementation avoids "brutish programming".

    - The condition is respected. The code is well written avoiding brutish programming.

45. Check order of computation/evaluation, operator precedence and parenthesizing.

    - This condition is respected in this method.
    - NOTE: useless parenthesis are used at line 1977.

46. Check the liberal use of parenthesis is used to avoid operator precedence problems.

    - The parenthesis are used correctly in the declarations and comparisons.

47. Check that all denominators of a division are prevented from being zero.

- There aren't any division.

48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.

    - This condition is respected in this method because there aren't arithmetic operations.

49. Check that the comparison and Boolean operators are correct.

    - This condition is respected in this method.

50. Check throw-catch expressions, and check that the error condition is actually legitimate.

    - This condition is respected in this method because there isn't any throw-catch expression.

51. Check that the code is free of any implicit type conversions.

    - Catching a generic Exception instance is not a good solution. It's easier to code but catching bugs will be harder in the future.
    - Catch Exception at line 1966 is not a good solution

Exceptions

52. Check that the relevant exceptions are caught.

    - All the possible exceptions are caught by the big try-catch block. As explained at point 51 catching a generic Exception is not a good solution. Creating an Exception class that can encapsulate the different kinds of thrown exceptions and check for it could be a better solution.

53. Check that the appropriate action are taken for each catch block.

    - An unchecked exception is thrown in the catch block. Probably a checked exception whould have been better because the comment at the head of the method ask to wrap the `preInvoke` method in a try catch block.

Flow of Control

54. In a switch statement, check that all cases are addressed by break or return.

    - This condition is respected in this method because there is no switch statement.

55. Check that all switch statements have a default branch.

    - This condition is respected in this method because there is no switch statement.

56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

    - This condition is respected in this method because there are no loops.

Files

No files are used in this method.

4 Other problems found

1. We noticed that the class EJBInstance could be substituted by a proper interface for the sake of encapsulation and information hiding and the advantages related to Object-oriented programming. In fact, methods analyzed used EJBInstance accessing to public fields. This is not a best practice for the Object-Oriented paradigm. Also the paradigm "`Code with interfaces`" should be used, but it's not.