Code Inspection
Version 1.0

Losio Davide Francesco, Luchetti Mauro, Mosca Paolo

January 5, 2016

Contents

1 — Classes and methods inspected

The following methods from the CurrentTransaction class are here analyzed:

- Name: numActive(Long localTID, boolean[] outstanding)
  - Start Line: 567
  - Location: appserver/transaction/jts/src/main/java/com/sun/jts/CosTransactions/CurrentTransaction.java
- Name: getSuspendedTransactions()
  - Start Line: 636
  - Location: appserver/transaction/jts/src/main/java/com/sun/jts/CosTransactions/CurrentTransaction.java
- Name: getRunningTransactions()
  - Start Line: 670
  - Location: appserver/transaction/jts/src/main/java/com/sun/jts/CosTransactions/CurrentTransaction.java
- Name: getAllTransactions()
  - Start Line: 703
  - Location: appserver/transaction/jts/src/main/java/com/sun/jts/CosTransactions/CurrentTransaction.java
- Name: sendingRequest(int id, PropagationContextHolder holder)
  - Start Line: 750
  - Location: appserver/transaction/jts/src/main/java/com/sun/jts/CosTransactions/CurrentTransaction.java

- Name: receivedReply (int id, PropagationContext context, org.omg.CORBA.Environment ex)
  - Start Line: 853
  - Location: appserver/transaction/jts/src/main/java/com/sun/jts/CosTransactions/CurrentTransaction.java
- Name: receivedRequest(int id, PropagationContext context)
  - Start Line: 971
  - Location: appserver/transaction/jts/src/main/java/com/sun/jts/CosTransactions/CurrentTransaction.java

2 — Functional role of assigned set of classes:

The class CurrentTransaction is used to manage the association of transactions with threads in a process, and associated state/operations. To do this, according to the methods Javadoc, the class:

- Sets up the thread association for a Control object.

  This is done following this condition: if there is a current Control object, and the method has not been asked to stack it, return FALSE to indicate that it cannot replace the object. Otherwise stack the current Control object behind the new one, which becomes the current Control for the thread. Otherwise, if there is no current Control, then just make the new one current.

- Removes the association for the thread under which the operation was invoked.

  First the method ensures that the current thread association is valid. Then, if there is a current Control, removes its association. If it is asked to unstack, the method gets the stacked Control, if any. If there is one, sets up the thread association.

- Ensures that an association with an aborted transaction is dealt with cleanly, this handling the aborted transactions (i.e. replaces the association by one with the first ancestor that has not been aborted, if any).

- Can either add or remove Control obj (ControlImpl) to/from a specific set in which there are all the suspended ones.

- Can return, if any, the current Control obj used by a thread.

- Can achieve the Coordinator obj that is the active thread.

- Can estimate the number of thread associations currently active for a given transaction.

- Can register or release a StaticResourse object making it able/unable to know whenever any association of a transaction with a tread is either started or ended.

- Can display all the transactions in the system that are currently running or suspended (jointly or singly) in the form of a sequence of Control objects.
- Handles the requests and the replies for the transactions.
- Can be informed whenever a reply or a request is sent.
- Can recreate a transaction starting from the information contained in the ID associating it to the current thread.
- Can disassociate the current thread of control from a specified transaction.
- Can end all thread associations for a given transaction or end a specific thread.
- Can be informed when the transaction service is being shut down.

For what concern the assigned methods, below are reported the respective JavaDoc documentation. This is done to show the role of each. Thanks to their clarity there is no need to add other explanations.

2.1 static int numActive(Long localTID, boolean[] outstanding)

JavaDoc documentation

```
/**Returns the number of thread associations currently active for the given
 * transaction identifier.
 * <p>
 * A boolean value indicating whether there are outstanding requests is returned
 * as an output parameter.
 * @param localTID The local transaction identifier.
 * @param outstanding  A 1-element array which will indicate outstanding requests.
 * @return The number of active thread associations.
 * @see
 */
```

Step by step explanation

```
static int numActive(Long localTID,boolean[/*1*/]outstanding){

  if(!statsOn){
    throw new NO_IMPLEMENT("statistics not on"); //check on status.
  }

    int result = 0; //variables initialization.
    outstanding[0] = false; //variables initialization.
    StatusHolder outStatus = new StatusHolder(); //variables initialization.

    //First check whether there are any outstanding requests.
    //Count all of the Control objects that have the same local TID
    //as that given.

    //Declaration and initialization of an iterator element, on
    //controlImpl elements.

    Enumeration controls = threadContexts.elements();

    //while on controls elements, it runs all the elements
    while(controls.hasMoreElements()) {

      //stores the current threads
      ControlImpl current = (ControlImpl)controls.nextElement();
```

```
      //If the Control object represents a transaction that
      //has been completed, don't count it.

      //get the transaction identifier
      outStatus.value = Status.StatusRolledBack;
        try {

          //get the thread associated
          Long currentLocalTID = current.getLocalTID(outStatus);

          //further checks on the transaction to get the correct
          //identifier
          if(outStatus.value == Status.StatusActive)

            //further checks on the threads to get the correct
            //threads
            if(currentLocalTID.equals(localTID)){

              //if it is outgoing or outstanding, puts the True
              //Boolean value into outstanding out parameter.
              outstanding[0] |= current.isOutgoing();

              //if the thread is associated to the transaction
              //count him
              result++;
            }
        } catch( Throwable exc ) {
        //error catching related probably to current.getLocalTID
        //(outStatus)
        _logger.log(Level.FINE,"", exc);
              }
    }
        return result; //return
}
```

2.2 static Control[] getSuspendedTransactions()

JavaDoc documentation

```
  /**Returns all the transactions in the system that are currently
    suspended
  * in the form of a sequence of Control objects.
  * @param
  * @return  The list of suspended Control objects.
  * @see
  */
```

Step by step explanation

```
static Control[] getSuspendedTransactions() {
    //check on status
    if(!statsOn){
      throw new NO_IMPLEMENT("statistics not on");
    }

    //Variable initialization and declaration
    Control[] result = null;
```

```
    //Copy the contents of the suspended set into the array.

    //check the presence of suspended transaction.
    int suspNum = suspended != null ? suspended.size() : 0;

    //perform only if there are suspended transaction
    if( suspNum > 0 ) {

      //initialize correctly the return variables. It is done here
      //because only here it is possible to know the right
      //size of the array
      result = new Control[suspNum];

      //Declaration and initialization of an iterator element, on
      //controlImpl elements.
      Enumeration controls = suspended.elements();

      //Variable declaration and initialization. Pos is used as
      //index of the result array
      int pos = 0;

      //iterate on the suspended threads individuated.
      while(controls.hasMoreElements())

        //put the threads into the return variable
        result[pos++] = ((ControlImpl)controls.nextElement())
                        .object();
    }

    //if no threads are individuated, return an empty vector.
    else
      result = new Control[0];

    //return statements
    return result;
}
```

2.3 static Control[] getRunningTransactions()

JavaDoc documentation:

```
  /**Returns all the transactions in the system that are currently
    running
  * (i.e. not suspended) in the form of a sequence of Control
    objects.
  * @param
  * @return  The list of running Control objects.
  * @see
  */
```

Step by step explanation

```
static Control[] getRunningTransactions() {
    if(!statsOn){ //check on status
      throw new NO_IMPLEMENT("statistics not on");
    }

    //Variable initialization and declaration
    Control[] result = null;
```

```
    // Copy the Control objects which have thread associations into
    //the result.

    //check the presence of running transaction.
    int runNum = threadContexts != null ? threadContexts.size() :
    0;

    //perform only if there are running transaction
    if( runNum > 0 ) {

      //initialize correctly the return variables. It is done here
      //because only here it is possible to know the right size of
      //the array
      result = new Control[runNum];

      //Declaration and initialization of an iterator element,
      //on controlImpl elements.
      Enumeration controls = threadContexts.elements();

      //Variable declaration and initialization. Pos is used as
      //index of the result array
      int pos = 0;

      //iterate on the running threads individuated.
      while(controls.hasMoreElements())

        //put the threads into the return variable
        result[pos++] = ((ControlImpl)controls.nextElement()).
    object();
    }

    //if no threads are individuated, return an empty vector.
    else
      result = new Control[0];

    //return statements
    return result;
}
```

2.4 static Control[] getAllTransactions()

JavaDoc documentation:

```
  /**Returns all the transactions in the system that are currently
    running
   * or suspended in the form of a sequence of Control objects.
   * @param
   * @return  The list of all Control objects.
   * @see
   */
```

Step by step explanation

```
static Control[] getAllTransactions() {
    if(!statsOn){ //check on status
      throw new NO_IMPLEMENT("statistics not on");
    }
```

```
    //Variable initialization and declaration
    Control[] result = null;

    //Copy the Control objects which have thread associations into
    the result.

    //check the presence of any kind of transaction.
    int allNum = threadContexts != null ? threadContexts.size()+
    suspended.size() : 0;

    //perform only if there are transaction
    if( allNum > 0 ) {

      //initialize correctly the return variables. It is done here
      //because only here it is possible to know the right size
      //of the array
      result = new Control[allNum];

      //Copy the contents of the suspended set into the array.

      //Declaration and initialization of an iterator element,
      //on controlImpl elements.
      Enumeration controls = suspended.elements();

      //Variable declaration and initialization. Pos is used
      //as index of the result array
      int pos = 0;

      //iterate on the suspended threads individuated.
      while(controls.hasMoreElements())

        //put the threads into the return variable
        result[pos++] = ((ControlImpl)controls.nextElement())
                        .object();

      //Copy the Control objects which have thread associations
    into the result.

      //initialization of an iterator element, on controlImpl
    elements.
      controls = threadContexts.elements();

      //iterate on the running threads individuated.
      while(controls.hasMoreElements())

        //put the threads into the return variable
        result[pos++] = ((ControlImpl)controls.nextElement())
                        .object();
    }

    //if no threads are individuated, return an empty vector.
    else
      result = new Control[0];

    //return statements
    return result;
}
```

2.5 static void sendingRequest(int id, PropagationContextHolder holder)

JavaDoc Documentation:

```
/**Informs the CurrentTransaction that a request is being sent.
* <p>
* Returns the transaction context that should be established for
    the object in
* the remote process.
*
* @param id  The request identifier.
* @param holder  The completed context object.
*
* @return
*
* @exception TRANSACTION_ROLLEDBACK  The current transaction has
    been rolled back. The message should not be sent and
* TRANSACTION_ROLLEDBACK should be returned to the caller.
* @exception TRANSACTION_REQUIRED  There is no current transaction.
*
* @see
*/
```

Step by step explanation

```
static void sendingRequest(int id, PropagationContextHolder holder)
  throws TRANSACTION_ROLLEDBACK, TRANSACTION_REQUIRED

  //Ensure that the current Control onj is valid. Return
  //immediately if not

  //Variable initialization and declaration
  boolean[] outBoolean = new boolean[1];

  //Get the current thread identifier, and the corresponding
  //Control object if there is one.
  ControlImpl current = endAborted(outBoolean, false);

  //if there isn't any
  if( outBoolean[0] ) {
    TRANSACTION_ROLLEDBACK exc = new TRANSACTION_ROLLEDBACK(0,
        CompletionStatus.COMPLETED_NO);
    throw exc;
  }

  //When there was no current transaction this part throwed the
  //TRANSACTION_REQUIRED exception. After the code modification
  //it simply returns the method without doing nothing.
  if( current == null ) {
      //$   TRANSACTION_REQUIRED exc = new TRANSACTION_REQUIRED();
      //$   if(trc != null) trc.event(EVT_THROW).data(exc).write();
      //$   throw exc;
    return;
  }
```

```
  try {
    // Get the the context from the Control object. If the context
    //is not available, then indicate that there is no transaction.
    holder.value = current.getTXContext();
  }

  //If the Coordinator is inactive, throw the INVALID_TRANSACTION
  //exception, as this will be because the transaction is not
  //able to do transactional work.
  catch (Unavailable exc) {
    INVALID_TRANSACTION ex2 = new INVALID_TRANSACTION(0,
        CompletionStatus.COMPLETED_NO);
    ex2.initCause(exc);
    throw ex2;
  }

  //If the Coordinator has rolled back, allow the
  //TRANSACTION_ROLLEDBACK exception, to pass to the caller.
  catch( TRANSACTION_ROLLEDBACK exc ) {

    //Stop the current Control obj.
    endCurrent(true);

    //Erase the current Control obj.
    current.destroy();

    throw (TRANSACTION_ROLLEDBACK)exc.fillInStackTrace();

  //Any other exception is unexpected. If it happens it is assumed
  //that there is no transaction.
  catch( Throwable exc ) {
    _logger.log(Level.FINE,"", exc);
  }
}
```

From the code reported above it could seem as the method does nothing outside checking exception, this because the part where the method "real acts" has been commented like below.

```

// COMMENT(Ram J) 11/25/2000 With the PI based OTS 1.2
// implementation, exception replies | location forwarded
// responses may not carry back a tx svc context (since
// the server OTS interceptor send point may not have been
// called. In such a case, it is impossible to enforce
// checked behaviour. The next revision of OTS 1.2 should
// address this, and provide a solution to the checked
// behaviour in a PI based OTS implementation. Then,
// these checks shall be enabled.

//current.incrementOutgoing();
```

2.6 static void receivedReply(int id, PropagationContext context, org.omg.CORBA.Environment ex)

JavaDoc Documentation:

```
/**Informs the CurrentTransaction that a reply has been received.
*
* @param id       The request identifier.
* @param context  The PropagationContext from the message.
* @param ex       The exception on the message.
*
* @return
*
* @exception WrongTransaction  The context returned on the reply is for a
*   different transaction from the current one on the thread.
*
* @see
*/
```

Step by step explanation

```
static void receivedReply(int id, PropagationContext context,
                org.omg.CORBA.Environment ex)
  throws org.omg.CORBA.WrongTransaction {

  //Look up the current Control object.
  //Thread thread = Thread.currentThread();
  ControlImpl current = (ControlImpl)m_tid.get();

  //If there is no current transaction, or an exception was raised,
  //then just return.
  if( current == null ) {
    return;
  }

  //If there is an active exception, report it.
  //OMG OTS issue 1819, if there is a system exception mark
  //the transaction for rollback
  java.lang.Exception ctxExc = ex.exception();

  if (ctxExc instanceof SystemException) {

    //Instance of a Coord instantiated as null.
    Coordinator currentCoord = null;

    //Try to get the coorect coordinator
    try {
      if (Configuration.isLocalFactory()) {
        currentCoord = current.get_localCoordinator();
      } else {
        currentCoord = current.get_coordinator();
      }
    } catch (Unavailable exc) {
      _logger.log(Level.FINE,"", exc);
    }
```

```
    //Without a coordinator, the transation cannot be mark
    //for rollback. Then return.
    if (currentCoord == null) {
      return;
    }

    //If the coordinator exist, then try to rollback
    try {
      currentCoord.rollback_only();
    } catch (Inactive exc) {
      _logger.log(Level.FINE,"", exc);
    }

    // COMMENT (Ram J) (11/24/2000) This has been commented out
    //since the exception reply could have a tx context.
    //Do further checks.
    //return;
  }

  // Return if there is no context on the message.
  if( context == null ||
    context.current == null ||
    context.current.coord == null ||
    context.current.otid.formatID == -1 ) {
      return;
  }

  //Get the global id from the current context. If the transaction
  //is not active, then end the current association.

  //Variables initialization and declaration
  StatusHolder outStatus = new StatusHolder();
  outStatus.value = Status.StatusRolledBack;
  GlobalTID globalTID = null;

  try {
    globalTID = new GlobalTID(current.getGlobalTID(outStatus));
  } catch( Throwable exc ) {
    _logger.log(Level.FINE,"", exc);
  }

  //If the global identifier is NULL, then the Control object
  //is unable to provide us with checking behaviour. We do not
  //check in this case.
  if( globalTID != null ) {

    //If the status is not active, end the current thread
    //association
    if( outStatus.value != Status.StatusActive ) {
      endCurrent(true);
      current.destroy();
      //org.omg.CORBA.WrongTransaction exc = new org.omg.CORBA
      //.WrongTransaction(0,CompletionStatus.COMPLETED_YES);
      org.omg.CORBA.WrongTransaction exc = new org.omg.CORBA
          .WrongTransaction();
      throw exc;
    }
```

```
    //If the global id is different from the one in the context,
    //then raise the org.omg.CORBA.WrongTransaction exception.
    if( !globalTID.isSameTID(context.current.otid) ) {
      //org.omg.CORBA.WrongTransaction exc = new org.omg.CORBA
      //.WrongTransaction(0,CompletionStatus.COMPLETED_YES);
      org.omg.CORBA.WrongTransaction exc = new org.omg.CORBA
          .WrongTransaction();
      throw exc;
    }
  }

  //If the Control object is not a proxy, then decrement the
  //outgoing count.

  //COMMENT(Ram J) 11/25/2000 With the PI based OTS 1.2
  //implementation, exception replies | location forwarded
  //responses may not carry back a tx svc context
  //(since the server OTS interceptor send point may not
  //have been called. In such a case, it is impossible to
  //enforce checked behaviour. The next revision
  //of OTS 1.2 should address this, and provide a solution
  //to the checked behaviour in a PI based OTS implementation.
  //Then, these checks shall be enabled.

  //current.decrementOutgoing();
}
```

2.7 static void receivedRequest(int id, PropagationContext context)

JavaDoc Documentation:

```
/**Informs the CurrentTransaction that a request has been received.
* <p>
* The request contains the transaction context that should be
    established
* for the object.
*
* @param id       The request identifier.
* @param context  The PropagationContext from the message.
*
* @return
*
* @see
*/
```

Step by step explanation

```
static void receivedRequest(int id, PropagationContext context) {

  //Return if there is no context on the message.
  //If the transaction identifier in the context is NULL, just
  //return.
  if( context == null ||
    context.current == null ||
    context.current.otid.formatID == -1 ) {
      return;
  }
```

```
11
12    //Init TransactionManagerImpl in the JTS delegate, so that
13    //all calls from this point forward use it correctly.
14    JavaEETransactionManagerJTSDelegate.getInstance().initXA();
15
16    //This part uses a local factory to recreate the transaction
17    //locally. Part of the code has been commenteted out.
18
19    //if( localFactory == null )
20    //localFactory = Configuration.getFactory();
21    //Control current = localFactory.recreate(context);
22    Control current = Configuration.getFactory().recreate(context);
23
24    //Record the imported transaction.
25    importedTransactions.put(Thread.currentThread(),
26                  new GlobalTID(context.current.otid));
27
28    //Create a new Control and associate it with the thread
29    try {
30        ControlImpl contImpl = null;
31        if (Configuration.isLocalFactory()) {
32          contImpl = (ControlImpl) current;
33        } else {
34          contImpl = ControlImpl.servant(JControlHelper
35                        .narrow(current));
36        }
37        setCurrent(contImpl,false);
38      }
39
40    //If any exception was thrown during that lot, then we have
41    //failed to create a subordinate. Do something drastic.
42    catch( Throwable exc ) {
43      _logger.log(Level.WARNING,"jts
44        .unable_to_create_subordinate_coordinator", exc);
45
46      Stringmsg = LogFormatter.getLocalizedMessage(_logger, "jts
47        .unable_to_create_subordinate_coordinator");
48
49      throw  new org.omg.CORBA.INTERNAL(msg);
50    }
51  }
```

3 — List of issues found by applying the checklist:

3.1 General Class Inspection

Class and Interface Declarations

- Variable "statsOn" at line 102, is not private. According to what stated at point 25 in the D) sub-clause it has to be moved on top of the private variables.
- Variable "_logger" at line 122, is not private. According to what stated at point 25 in the D) sub-clause it has to be moved on top of the private variables.
- As stated in point 27, CurrentTransaction class is definitely too big. It has too many lines, about 1400, and this makes really difficult the ordinary errors checks.

3.2 static int numActive (Long localTID, boolean[] outstanding)

Naming Conventions

- In line 570 the exception name is in a wrong format. This format convention must be used with constants and not with class names, as stated at point 7.
- Variable name "result" at line 573 might be a too generic name.
- Variable name "StatusActive" at line 590 start with capital S, this is wrong according to what stated at point 6.
- Variable name "outstanding" at line 574 might be a too generic name.
- Variable name "controls" at line 580 might be a too generic name.
- Method name "numActive" is not a verb as stated at point 5.

Indention

- Line 569, 570, 571 and 596 of the method, make use of tab instead of space as stated at point 9.

Braces

- "If" statement at line 590 doesn't have curly braces, as stated in point 11.

File Organization

- Might be useful to have a blank line between 569 line and 568 line, as method declaration and method code are two different section as stated in point 12.
- Blank lines at 572 and 599 can be deleted, as stated in point 12.
- Comments at lines 578 and 584 are too long, they have more than 80 characters. A line break might be considered as stated at point 13.

Wrapping Lines

- Line break at line 567 is avoidable, since the line is shorter than 80 characters and there is nor comma or operator needing the line break. This is stated at line 15. It should be also noted that at the same line is visible an improper and meaningless use of blank space.

Comments

- Comments at line 584/585 are a bit confusing, the code stated below, indeed, seems to be more focused on counting something (result++) than to not count. This is stated at point 18.
- Comments at line 577/578 aren't stated in a right position, indeed below is stated a declaration. They have to be moved below the declaration statement.
- Comments "/*1*/" is probably now useless. If is it so, it has to be deleted as stated at point 19.

Class and Interface Declarations

- Access to "Status.StatusRolledBack" at line 587, might be done by the use of getters and setters, to avoid encapsulation breaking as stated at point 27. although, since this seems to be a library class, this cold be considered as an unavoidable concession.
- The "getSuspendedTransactions()" methods is not grouped by functionality. It might be out of place because it is a "getter" method, hence, might be better to place it at the end of the class.

Initialization and declarations

- Variable "controls" at line 580, has to be moved upon comments at line 577 and 578, as stated at point 33.
- Variable "current" at line 582 is declared in a wrong place, as stated at point 33.
- Variable "currentLocalTID" at line 589 is declared in a wrong place, as stated at point 33.

Arrays

- There are no checks on the initialization of "outstanding" array given as parameter to the method. So we actually don't know if the access to this array at line 574 is consistent. This is requested at point 39.

Object Comparison

- If statement, at line 590 is comparing two Status objects, "StatusRolled-Back" thorough "outStatus.value" and "Status.StatusActive". All of this by the using of "==" operator. Probably is not a good thing as stated at point 40. It might be correct use the "equals" operator.

Output Format

- Catch clause at line 595 hasn't a good error message accordingly to point 42. Instead could be better to specify the exact problem thrown. Since the exception is thrown by the 589 line statement, it might be better to catch a "SystemException ". This is the exception thrown by "current.getLocalTID(outStatus)".

Computation, Comparison and Assignments

- Catch clause at line 595 hasn't a good formulation of the error condition. It is indeed probably too generalized. Instead could be better to specify the exact problem that could be thrown. Since the exception is thrown by the 589 line statement, it might be better to catch a "SystemException ". Indeed this is the exception thrown by "current.getLocalTID(outStatus)". This also accordingly to point 42 of the Output Format section.
- The use of a Boolean array of one-element is probably a case of "brutish programming". Actually the case is not clearly stated at point 44, but we don't see the utility of using a one-element array instead of a simpler Boolean variable. Since this is probably done to return "outStanding" array as an output parameter, it probably might be better to return an object that embedded into itself both result and outStanding boolean value.

Exception

- Catch clause at line 595 only print the error message on the terminal. According to point 50, this is probably not the best solution to the error handling.

3.3 static Control[] getSuspendedTransactions ()

Naming Conventions

- In line 639 the exception name is in a wrong format. This format convention must be used with constants and not with class names, as stated at point 7.
- Variable name "result" at line 642 might be a too generic name.

Indention

- Line 638, 639 and 640 of the method, make use of tab instead of space as stated at point 9.
- Indention at line 639 make use of useless blank space characters, this is against point 8.

Braces

- "While" statement at line 652 doesn't have curly braces, as stated in point 11.
- "Else" statement at line 655 doesn't have curly braces, as stated in point 11.

File Organization

- Blank 649 line might be useless, as stated in point 12. Instead could be useful a blank line to separate the if statements to the while statements at line 652.

Comments

- Comments at line 644 is out of position. It might be better to move it between line 651 and 652. Furthermore is a bit useless as the code is quite self-explaining. This rule is stated at point 18.

Initialization and declarations

- Variable declaration "result" at line 642, has to be moved on top of the block, as stated at point 33. There's also an useless initialization. The value "null" assigned here goes against point 32. The value of the variable depends upon the computation and in this case the variable can be not inizialized.

- Variable declaration “supsNum” at line 646, has to be moved on top of the block, as stated at point 33.
- Variable declaration “controls” at line 650, has to be moved on top of the block, as stated at point 33.
- Variable declaration “pos” at line 651, has to be moved on top of the block, as stated at point 33.

Arrays

- “Result” array constructor at line 640, is not correctly called. Instead it is assigned a null value. This is against point 39.

Computation, Comparisons and Assignments

- Type conversions at line 653 are quite complicate to understand. Although they are declared in comment as well as explained in javadoc, it is advisable to split them in a more clear sequence.

Flow of Control

- As stated at point 56, it is advisable to have a deeper control statement in line 652 while. In this while it is only checked the length of suspended number of elements thorough the use of controls variable. It is true that the size of result array appears to be the same of the controls variable size, as suggested by lines 646 and 650, but a greater attention to these details could avoid silly and boring mistakes. Might be useful to threat explicitly in the code the “outOfBoundException” instead to treat it implicitly in the code as stated at point 52.

3.4 static Control[] getRunningTransactions ()

Naming Conventions

- In line 672 the exception name is in a wrong format. This format convention must be used with constants and not with class names, as stated at point 7.
- Variable name “result” at line 675 might be a too generic name.

Indention

- Line 671, 672 and 673 of the method, make use of tab instead of space as stated at point 9.
- Indention at line 672 make use of useless blank space characters, this is against point 8.

Braces

- "While" statement at line 685 doesn't have curly braces, as stated in point 11.
- "Else" statement at line 688 doesn't have curly braces, as stated in point 11.

File Organization

- Blank 682 line might be useless, as stated in point 12. Instead could be useful a blank line to separate the if statements to the while statements at line 685.

Comments

- Comments at line 677 is out of position. It might be better to move it between line 684 and 685. Furthermore is a bit useless as the code is quite self-explaining. This rule is stated at point 18.

Initialization and declarations

- Variable declaration "result" at line 675, has to be moved on top of the block, as stated at point 33. There's also an useless initialization. The value "null" assigned here goes against point 32. The value of the variable depends upon the computation and in this case the variable can be not inizialized.
- Variable declaration "runNum" at line 679, has to be moved on top of the block, as stated at point 33.
- Variable declaration "controls" at line 683, has to be moved on top of the block, as stated at point 33. It is suggested to split the -declaration- to -declaration and assignment- as stated at point 32.
- Variable declaration "pos" at line 684, has to be moved on top of the block, as stated at point 33.

Arrays

- "Result" array constructor at line 675, is not correctly called. Instead it is assigned a null value. This is against point 39.

Computation, Comparisons and Assignments

- Type conversions at line 686 are quite complicate to understand. although they are declared in comment as well as explained in javadoc, it is advisable to split them in a more clear sequence.

Flow of Control

- As stated at point 56, it is advisable to have a deeper control statement in line 685. In this while it is only checked the length of running number of elements thorough the use of controls variable. It is true that the size of result array appears to be the same of the controls variable size, as suggested by lines 679 and 683, but a greater attention to these details could avoid silly and boring mistakes. Might be useful to threat explicitly in the code the "outOfBoundException" instead to treat it implicitly in the code as stated at point 52.

3.5 static Control[] getAllTransactions ()

Naming Conventions

- In line 706 the exception name is in a wrong format. This format convention must be used with constants and not with class names, as stated at point 7.
- Variable name "result" at line 708 might be a too generic name.

Indention

- Line 705, 706 and 707 of the method, make use of tab instead of space as stated at point 9.
- Indention at line 706 make use of useless blank space characters, this is against point 8.

Braces

- "While" statement at line 718 doesn't have curly braces, as stated in point 11.
- "While" statement at line 724 doesn't have curly braces, as stated in point 11.
- "Else" statement at line 727 doesn't have curly braces, as stated in point 11.

File Organization

- Blank 704 and 709 lines might be useless, as stated in point 12. Instead blank lines could be useful at lines 707,711,717 and 724. This in order to split up the logical differences. -While- blocks have to be separated from the other statements and -variables declaration- from code statements. Although, in any case, variables remaining declared in wrong places.
- Line 710 is too long, it has more than 80 characters. A line break might be considered as stated at point 13.

Comments

- Comments at line 714 is out of position. It might be better to move it between line 717 and 718. Furthermore is a bit useless as the code is quite self-explaining. This rule is stated at point 18.
- Comments at line 721 is out of position. It might be better to move it between line 723 and 724. Furthermore is a bit useless as the code is quite self-explaining. This rules is stated at point 18.

Initialization and declarations

- Variable declaration "result" at line 708, has to be moved on top of the block, as stated at point 33. There's also an useless initialization. The value "null" assigned here goes against point 32. The value of the variable depends upon the computation and in this case the variable can be not inizialized.
- Variable declaration "allNum" at line 710, has to be moved on top of the block, as stated at point 33.
- Variable declaration "controls" at line 716, has to be moved on top of the block, as stated at point 33. It is suggested to split the -declaration- to -declaration and assignment- as stated at point 32.
- Variable declaration "pos" at line 717, has to be moved on top of the block, as stated at point 33.

Arrays

- "Result" array constructor at line 708, is not correctly called. Instead it is assigned a null value. This is against point 39.

Computation, Comparisons and Assignments

- Type conversions at lines 719 and 725 are quite complicate to understand. Although they are declared in comment as well as explained in javadoc, it is advisable to split them in a more clear sequence.
- The code block going from line 712 to line 725 present several duplicate statements. As stated at point 44 this can be considered as brute force solution. It is advisable to re-organize the code and trying to figure-out a smarter and simpler solution.

Flow of Control

- As stated at point 56, it is advisable to have a deeper control statement in lines 718 and 724. In these while blocks it is only checked the length of number of elements thorough the use of controls variable. It is true that the size of result array appears to be the same of the controls variable size, as suggested by line 710, 716 and 723 (716 and 723, summed thorough the two cycles gave exactly the 710 size), but a greater attention to these details could avoid silly and boring mistakes. Especially in these complicates situations. Might be useful to threat explicitly in the code the "outOfBoundException" instead to treat it implicitly in the code as stated at point 5.

3.6 static void sendingRequest (int id, PropagationContextHolder holder)

Naming Conventions

- Parameter name "id" at line 750 might be a too generic name.
- In line 752 the two exception names are in a wrong format. This format convention must be used with constants and not with class names, as stated at point 7.
- The class name should be more meaningful, in fact the class after done some checks, should(cause this part is a commented) call "incrementOutgoing()" that increments the incomplete asynchronous request counter (from JavaDoc). So for example a name like "IncrementOutgoingReq" would have been better.

Indention

- Line 824 of the method makes use of tab instead of space as stated at point 9.

Braces

- "If" statement at line 768 doesn't have curly braces, as stated in point 11. This part is commented but it is wanted to highlight the error in case of future use.
- "If" statement at line 790 doesn't have curly braces, as stated in point 11. This part is commented but it is wanted to highlight the error in case of future use.

File Organization

- Lines 782 and 807 have more than 80 characters but they are two assignments. They don't exceed 120 characters so it may not be a problem.

Wrapping Lines

- Line break at line 750 is avoidable, since the line is shorter than 80 characters and there is nor comma or operator needing the line break. This is stated at line 15.
- It might be present a line break after the comma in line 782, as stated at point 15.
- It might be present a line break after the comma in line 807, as stated at point 15.

Comments

- Lines 761-774 have an explanation of why, and also a date of when, the code has been commented out. Otherwise, according to the point 19, should be written the date it can be removed from the source file if determined it is no longer needed.
- Comment in line 786 is no longer consistent with the code changes applied by commenting out part of code.
- Lines 830-837 have an explanation of why, and also a date of when, the code has been commented out. Otherwise, according to the point 19, should be written the date it can be removed from the source file if determined it is no longer needed.

Initialization and Declarations

- Variable declaration "current" at line 780, has to be moved on top of the block, as stated at point 33.

Arrays

- In line 780 the value of outBoolean[0] is assigned by the method call "endAborted" and not during the declaration of the variable outBoolean[]. Changing the method behaviour can cause an error during the "if" check in line 781.

3.7 static void receivedReply (int id, PropagationContext context, org.omg.CORBA.Environment ex)

Naming Conventions

- Parameter names "id" and "ex" at line 583 might be a too generic names.
- Variable named "ctxExc" at line 874 might be a not meaningful name.
- Variable named "exc" at line 942 might be a not meaningful name.

Indention

- Line 885 and 895 of the method, make use of tab instead of space as stated at point 9.

File Organization

- Blank line at 876 might be useless, as stated in point 12.
- Lines 933 and 942 have more than 80 characters but they are two assignments. They don't exceed 120 characters so it may not be a problem.

Wrapping Lines

- Line break at line 853 is avoidable, since the line is shorter than 80 characters and there is nor comma or operator needing the line break. This is stated at line 15.

Comments

- Code commented at line 860 has no explanation of why, and no date of when, it can be removed.
- Code commented at line 932 has no explanation of why, and no date of when, it can be removed.
- Code commented at line 941 has no explanation of why, and no date of when, it can be removed.
- Lines 898-900 and 949-956 have an explanation of why, and also a date of when, the code has been commented out. Otherwise, according to the point 19, should be written the date it can be removed from the source file if determined it is no longer needed.

Initialization and Declarations

- Variable declaration "currentCoord" at line 877 has an useless initialization. The value "null" assigned here goes against point 32. The value of the variable depends upon the computation and in this case the variable can be not inizialized.
- Variable declaration "outStatus" at line 915, has to be moved on top of the block, as stated at point 33.
- Variable declaration "globalTID" at line 917, has to be moved on top of the block, as stated at point 33.

3.8 static void receivedRequest (int id, PropagationContext context)

Naming Conventions

- Parameter name "id" at line 971 might be a too generic name.

Indention

- Lines from 1014 to 1017 of the method, make use of tab instead of space as stated at point 9.

Braces

- "If" statement at line 989 doesn't have curly braces, as stated in point 11. This part is commented but it is wanted to highlight the error in case of future use.

File Organization

- Line 996 has more than 80 characters but it is a method call. It doesn't exceed 120 characters so it may not be a problem.
- Line 1014 has more than 80 characters but it is a method call. It doesn't exceed 120 characters so it may not be a problem.
- Line 1016 is too long, it has more than 120 characters. A line break might be considered as stated at point 13.

Wrapping Lines

- Line break at line 971 is avoidable, since the line is shorter than 80 characters and there is nor comma or operator needing the line break. This is stated at line 15.
- It might be present a line break after the comma in line 996, as stated at point 15.

Comments

- Code commented at line 989-991 has no explanation of why, and no date of when, it can be removed.

Initialization and Declarations

- Variable declaration "current" at line 992 has to be moved on top of the block, as stated at point 33.

4 — Appendix

4.1 Used Tools

We used the following tools to make the Code Inspection document:

- LYX 2.1: to redact and format the document;
- NetBeans IDE 8.1: to analyze and inspect the source code provided.

4.2 Time Spent

- For redact, correct and review this document we spent almost 18 hours per person.