Politecnico di Milano

A.A. 2015-2016
Software Engineering 2

CODE INSPECTION ASSIGNMENT

Alessandro Macchi
Caterina Finetti
Simone Manzoli

1: Assigned Methods

- commit()
- rollback()
- toString()

Location:
appserver/transaction/jta/src/main/java/com/sun/enterprise/transaction/JavaEETransactionImpl.java

2: Functional role of the classes

The entire class is an extension of a transaction object that already exists in the Transaction Interface, but optimized for some local transactions that need also a timer (timeout exceptions); in fact, all the non-local transactions are delegated to other classes. So we concluded that the commit() method is the one which complete the requested transaction that is represented by the object, exactly as specified in the JAVA documentation for the interface, but with a timer added; if the timeout happens during the transaction, it will be rolled back.

3: List of issues found by applying the checklist

We are now exposing the problems we found in the code that has been assigned to us. The red lines are ours comments to the code.

3.1: commit()

Statement 5: In our opinion, the "rollback()" method should be called "rollBack()", because it is a composite verb and "back" is actually a single word. So, this mistake extends to RollbackException, and appears in this method in lines 390, 391, 434, 436, 439, 442, 445, 455, 471, 483, 485, 488, 494, 525.

Statement 12: We found some unnecessary blank lines in the code. In particular, lines 393, 406, 427, 438, 443, 447, 462, 464, 478, 510, 514, 519, 524, 528, 531, 540 should be removed.

Statement 19: there are some comments that should have been removed from the code, as they are clearly some kind of memorandum for the developers, in lines 521, 526, 530, 536.

From now on, we are going to paste the lines where we found mistakes.

```
398   if (isTimerTask) /*statement 11, no braces for the "if"*/
        cancelTimerTask();

408   boolean success = false; /*statement 33, the declaration is not at the
beginning of the block */
                    if ( jtsTx != null ) {/*statement 18, this block should be
explained by comments */
410                     try {
                           ..........  /*this part should be commented*/
428                 } else { // local tx


433    if ( nonXAResource != null )  /*statement 11, no braces for the "if"*/
                  nonXAResource.getXAResource().rollback(xid);

 441    if ( nonXAResource != null ) /*statement 11, no braces for the "if"*/

459     _logger.log(Level.WARNING,
"enterprise_distributedtx.before_completion_excep", ex);
              // XXX-V2 no setRollbackOnly() ???
        /*statement 18 and/or 19, this comment is unnecessary or not clear*/
```

```
486  if(jtsTx == null) {  /*statement 40, the operator "==" should be replaced
by an "equal"*/
            if ( nonXAResource != null ) /*statement 11, no braces for the "if"*/
                    nonXAResource.getXAResource().rollback(xid);

517   if ( nonXAResource != null ) /*statement 11, no braces for the "if"*/

537    localTxStatus = Status.STATUS_ROLLEDBACK; // V2-XXX is this correct ?
              SystemException exc = new SystemException();
        /*statement 33, the declaration is not at the beginning of the block */

541   } finally {/*statement 18, this block should be explained by comments */
```

3.2: rollback()

Statement 5: In our opinion, the "rollback()" method should be called "rollBack()", because it is a composite verb and "back" is actually a single word.

Statement 12: We found some unnecessary blank lines in the code. In particular, lines 581, 588, 592, 594, 604, 625, 627 should be removed.

Statement 19: there are some comments that should have been removed from the code, as they are clearly some kind of memorandum for the developers, in line 602.

From now on, we are going to paste the lines where we found mistakes.

```
573   if (isTimerTask)   /*statement 11, no braces for the "if"*/
                  cancelTimerTask();

577  if (_logger.isLoggable(Level.FINE)){  /*statement 11, no braces for the
if"*/
           _logger.log(Level.FINE,"--In JavaEETransactionImpl.rollback,
         jtsTx="+jtsTx
         +" nonXAResource="+nonXAResource); /* statement 15, missing spaces
after operators */

582  if ( jtsTx == null )    /*statement 11, no braces for the "if"*/   /*statement
40, the operator "==" should be replaced by an "equal"*/
        checkTransationActive(); // non-xa transaction can't be in prepared state,
xa code will do its check

586  if ( jtsTx != null ) /*statement 11, no braces for the "if"*/
          jtsTx.rollback();

590  if ( nonXAResource != null ) /*statement 11, no braces for the "if"*/
                nonXAResource.getXAResource().rollback(xid);

607   if ( jtsTx == null ) {      /*statement 40, the operator "==" should be
replaced by an "equal"*/
          for ( int i=0; i<interposedSyncs.size(); i++ ) ){
/* statement 15, missing spaces after operators */

618  for ( int i=0; i<syncs.size(); i++ ) {
/*statement 40, the operator "==" should be replaced by an "equal"*/
```

3.3: toString()

Statement 12: We found some unnecessary blank lines in the code. In particular, lines 806, 812, 814, 822, 830 should be removed.
Statement 19: there are some comments that should have been removed from the code, as they are a second option to format the line, with the addiction of the square brackets. In the final implementation lines 808, 811, 823, 829 should be removed.

```
805  if( stringForm != null ) return stringForm;  /*statement 11, no braces for
the "if"*/

816  for( int i = 0; i < globalLen; i++ ) {
        int currCharHigh = (gtrId[i]&0xf0) >> 4;
        int currCharLow  = gtrId[i]&0x0f;
/* the declaration inside the cycle is avoidable; the variables should have been
declared outside and just reassigned in the cycle. */

823  //buff[pos++] = ':';
           buff[pos++] = '_';
/* statement 19: the ":" seems to be replaced by "_" but the comments doesn't
explain why (line 810 still report the ":" version) */
```

4: Code

In the previous pages, the lines we copied and pasted from the source code may have some additional line brake, in order to make the document easier to read; to complete the document, we attach below the complete methods without modifications.

```
390 public void commit() throws RollbackException,
                HeuristicMixedException, HeuristicRollbackException,
                SecurityException, IllegalStateException, SystemException {

        checkTransationActive();

        // START local transaction timeout
        // If this transaction is set for timeout, cancel it as it is in the commit state
        if (isTimerTask)
            cancelTimerTask();
400
        // END local transaction timeout
        if (_logger.isLoggable(Level.FINE)) {
            _logger.log(Level.FINE,"--In JavaEETransactionImpl.commit, jtsTx="+jtsTx
                +" nonXAResource="+ nonXAResource);
        }

        commitStarted = true;
        boolean success = false;
        if ( jtsTx != null ) {
410         try {
                jtsTx.commit();
                success = true;
            } catch(HeuristicMixedException e) {
                success = true;
                throw e;
            } finally {
                ((JavaEETransactionManagerSimplified) javaEETM).monitorTxCompleted(this, success);
                ((JavaEETransactionManagerSimplified) javaEETM).clearThreadTx();
                onTxCompletion(success);
420             try {
                    localTxStatus = jtsTx.getStatus();
                } catch (Exception e) {
                    localTxStatus = Status.STATUS_NO_TRANSACTION;
                }
                jtsTx = null;
            }

        } else { // local tx
            Exception caughtException = null;
430         try {
                if ( timedOut ) {
                    // rollback nonXA resource
                    if ( nonXAResource != null )
                        nonXAResource.getXAResource().rollback(xid);
                    localTxStatus = Status.STATUS_ROLLEDBACK;
                    throw new
RollbackException(sm.getString("enterprise_distributedtx.rollback_timeout"));
                }

                if ( isRollbackOnly() ) {
440                 // rollback nonXA resource
                    if ( nonXAResource != null )
                        nonXAResource.getXAResource().rollback(xid);

                    localTxStatus = Status.STATUS_ROLLEDBACK;
                    throw new
RollbackException(sm.getString("enterprise_distributedtx.mark_rollback"));
                }

                // call beforeCompletion
                for ( int i=0; i<syncs.size(); i++ ) {
```

```
450                 try {
                        Synchronization sync = (Synchronization)syncs.elementAt(i);
                        sync.beforeCompletion();
                    } catch ( RuntimeException ex ) {
                        _logger.log(Level.WARNING,
"enterprise_distributedtx.before_completion_excep", ex);
                        setRollbackOnly();
                        caughtException = ex;
                        break;
                    } catch (Exception ex) {
                        _logger.log(Level.WARNING,
"enterprise_distributedtx.before_completion_excep", ex);
460                     // XXX-V2 no setRollbackOnly() ???
                    }

                }

                for ( int i=0; i<interposedSyncs.size(); i++ ) {
                    try {
                        Synchronization sync = (Synchronization)interposedSyncs.elementAt(i);
                        sync.beforeCompletion();
                    } catch ( RuntimeException ex ) {
470                     _logger.log(Level.WARNING,
"enterprise_distributedtx.before_completion_excep", ex);
                        setRollbackOnly();
                        caughtException = ex;
                        break;
                    } catch (Exception ex) {
                        _logger.log(Level.WARNING,
"enterprise_distributedtx.before_completion_excep", ex);
                        // XXX-V2 no setRollbackOnly() ???
                    }

                }
480
                // check rollbackonly again, in case any of the beforeCompletion
                // calls marked it for rollback.
                if ( isRollbackOnly()) {
                    //Check if it is a Local Transaction
                    RollbackException rbe = null;
                    if(jtsTx == null) {
                        if ( nonXAResource != null )
                            nonXAResource.getXAResource().rollback(xid);
                        localTxStatus = Status.STATUS_ROLLEDBACK;
490                     rbe = new
RollbackException(sm.getString("enterprise_distributedtx.mark_rollback"));

                    // else it is a global transaction
                    } else {
                        jtsTx.rollback();
                        localTxStatus = Status.STATUS_ROLLEDBACK;
                        rbe = new
RollbackException(sm.getString("enterprise_distributedtx.mark_rollback"));
                    }

                    // RollbackException doesn't have a constructor that takes a Throwable.
500                 if (caughtException != null) {
                        rbe.initCause(caughtException);
                    }
                    throw rbe;
                }

                // check if there is a jtsTx active, in case any of the
                // beforeCompletions registered the first XA resource.
                if ( jtsTx != null ) {
                    jtsTx.commit();
510
                    // Note: JTS will not call afterCompletions in this case,
                    // because no syncs have been registered with JTS.
                    // So afterCompletions are called in finally block below.

                } else {
                    // do single-phase commit on nonXA resource
                    if ( nonXAResource != null )
                        nonXAResource.getXAResource().commit(xid, true);

520             }
                // V2-XXX should this be STATUS_NO_TRANSACTION ?
```

```
                localTxStatus = Status.STATUS_COMMITTED;
                success = true;

            } catch ( RollbackException ex ) {
                localTxStatus = Status.STATUS_ROLLEDBACK; // V2-XXX is this correct ?
                throw ex;

            } catch ( SystemException ex ) {
530             // localTxStatus = Status.STATUS_ROLLEDBACK; // V2-XXX is this correct ?
                localTxStatus = Status.STATUS_COMMITTING;
                success = true;
                throw ex;

            } catch ( Exception ex ) {
                localTxStatus = Status.STATUS_ROLLEDBACK; // V2-XXX is this correct ?
                SystemException exc = new SystemException();
                exc.initCause(ex);
                throw exc;
540
            } finally {
                ((JavaEETransactionManagerSimplified) javaEETM).monitorTxCompleted(this, success);
                ((JavaEETransactionManagerSimplified) javaEETM).clearThreadTx();
                for ( int i=0; i<interposedSyncs.size(); i++ ) {
                    try {
                        Synchronization sync = (Synchronization)interposedSyncs.elementAt(i);
                        sync.afterCompletion(localTxStatus);
                    } catch ( Exception ex ) {
                        _logger.log(Level.WARNING,
"enterprise_distributedtx.after_completion_excep", ex);
550                 }
                }

                // call afterCompletions
                for ( int i=0; i<syncs.size(); i++ ) {
                    try {
                        Synchronization sync = (Synchronization)syncs.elementAt(i);
                        sync.afterCompletion(localTxStatus);
                    } catch ( Exception ex ) {
                        _logger.log(Level.WARNING,
"enterprise_distributedtx.after_completion_excep", ex);
560                 }
                }

                onTxCompletion(success);
                jtsTx = null;
            }
        }
    }

    public void rollback() throws IllegalStateException, SystemException {
570
        // START local transaction timeout
        // If this transaction is set for timeout, cancel it as it is in the rollback state
        if (isTimerTask)
            cancelTimerTask();
        // END local transaction timeout

        if (_logger.isLoggable(Level.FINE)) {
            _logger.log(Level.FINE,"--In JavaEETransactionImpl.rollback, jtsTx="+jtsTx
                    +" nonXAResource="+nonXAResource);
580     }

        if ( jtsTx == null )
            checkTransationActive(); // non-xa transaction can't be in prepared state, xa code will
do its check

        try {
            if ( jtsTx != null )
                jtsTx.rollback();

            else { // rollback nonXA resource
590             if ( nonXAResource != null )
                    nonXAResource.getXAResource().rollback(xid);

            }

        } catch ( SystemException ex ) {
            throw ex;
```

```
        } catch ( IllegalStateException ex ) {
            throw ex;
        } catch ( Exception ex ) {
600         _logger.log(Level.WARNING, "enterprise_distributedtx.some_excep", ex);
        } finally {
            // V2-XXX should this be STATUS_NO_TRANSACTION ?
            localTxStatus = Status.STATUS_ROLLEDBACK;

            ((JavaEETransactionManagerSimplified) javaEETM).monitorTxCompleted(this, false);
            ((JavaEETransactionManagerSimplified) javaEETM).clearThreadTx();
            if ( jtsTx == null ) {
                for ( int i=0; i<interposedSyncs.size(); i++ ) {
                    try {
610                     Synchronization sync = (Synchronization)interposedSyncs.elementAt(i);
                        sync.afterCompletion(Status.STATUS_ROLLEDBACK);
                    } catch ( Exception ex ) {
                        _logger.log(Level.WARNING,
"enterprise_distributedtx.after_completion_excep", ex);
                    }
                }

                // call afterCompletions
                for ( int i=0; i<syncs.size(); i++ ) {
                    try {
620                     Synchronization sync = (Synchronization)syncs.elementAt(i);
                        sync.afterCompletion(Status.STATUS_ROLLEDBACK);
                    } catch ( Exception ex ) {
                        _logger.log(Level.WARNING,
"enterprise_distributedtx.after_completion_excep", ex);
                    }

                }

            }
            onTxCompletion(false);
630         jtsTx = null;
        }
    }

. . . . . .


801     public String toString(){

            // If we have a cached copy of the string form of the global identifier, return
            // it now.
            if( stringForm != null ) return stringForm;

            // Otherwise format the global identifier.
            //char[] buff = new char[gtrId.length*2 + 2/*'[' and ']'*/ + 3/*bqual and ':'*/];
            char[] buff = new char[gtrId.length*2 + 3/*bqual and ':'*/];
810         int pos = 0;
            //buff[pos++] = '[';

            // Convert the global transaction identifier into a string of hex digits.

            int globalLen = gtrId.length ;
            for( int i = 0; i < globalLen; i++ ) {
                int currCharHigh = (gtrId[i]&0xf0) >> 4;
                int currCharLow  = gtrId[i]&0x0f;
                buff[pos++] = (char)(currCharHigh + (currCharHigh > 9 ? 'A'-10 : '0'));
820             buff[pos++] = (char)(currCharLow  + (currCharLow  > 9 ? 'A'-10 : '0'));
            }

            //buff[pos++] = ':';
            buff[pos++] = '_';
            int currCharHigh = (0&0xf0) >> 4;
            int currCharLow  = 0&0x0f;
            buff[pos++] = (char)(currCharHigh + (currCharHigh > 9 ? 'A'-10 : '0'));
            buff[pos++] = (char)(currCharLow  + (currCharLow  > 9 ? 'A'-10 : '0'));
            //buff[pos] = ']';
830
            // Cache the string form of the global identifier.
            stringForm = new String(buff);

            return stringForm;
        }
```

5: Checklist

We attach the checklist of issues that we used.

Naming Conventions

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

Indention

8. Three or four spaces are used for indentation and done so consistently
9. No tabs are used to indent

Braces

10. Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block).
11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.

File Organization

12. Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).
13. Where practical, line length does not exceed 80 characters.
14. When line length must exceed 80 characters, it does NOT exceed 120 characters.

Wrapping Lines

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

Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.
19. Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.

Java Source Files

20. Each Java source file contains a single public class or interface.
21. The public class is the first class or interface in the file.
22. Check that the external program interfaces are implemented consistently with what is described in the javadoc.
23. Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

Package and Import Statements

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

Class and Interface 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
26. Methods are grouped by functionality rather than by scope or accessibility.
27. Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

Initialization and Declarations

28. Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected)
29. Check that variables are declared in the proper scope
30. Check that constructors are called when a new object is desired
31. Check that all object references are initialized before use
32. Variables are initialized where they are declared, unless dependent upon a computation
33. Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces "{" and "}"). The exception is a variable can be declared in a 'for' loop.

Method Calls

34. Check that parameters are presented in the correct order

35. Check that the correct method is being called, or should it be a different method with a similar name
36. Check that method returned values are used properly

Arrays

37. Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index)
38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds
39. Check that constructors are called when a new array item is desired

Object Comparison

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

Output Format

41. Check that displayed output is free of spelling and grammatical errors
42. Check that error messages are comprehensive and provide guidance as to how to correct the problem
43. Check that the output is formatted correctly in terms of line stepping and spacing

Computation, Comparisons and Assignments

44. Check that the implementation avoids “brutish programming: (see http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html)
45. Check order of computation/evaluation, operator precedence and parenthesizing
46. Check the liberal use of parenthesis is used to avoid operator precedence problems.
47. Check that all denominators of a division are prevented from being zero
48. Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding
49. Check that the comparison and Boolean operators are correct
50. Check throw-catch expressions, and check that the error condition is actually legitimate
51. Check that the code is free of any implicit type conversions

Exceptions

52. Check that the relevant exceptions are caught
53. Check that the appropriate action are taken for each catch block

Flow of Control

54. In a switch statement, check that all cases are addressed by break or return
55. Check that all switch statements have a default branch
56. Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions

Files

57. Check that all files are properly declared and opened
58. Check that all files are closed properly, even in the case of an error
59. Check that EOF conditions are detected and handled correctly
60. Check that all file exceptions are caught and dealt with accordingly