Politecnico di Milano
Academic Year 2015/2016
Software Engineering 2: "myTaxiService"
Code Inspection

Massimo Schiavo, Marco Edoardo Cittar

January 4, 2016

Contents

1 Assigned classes **3**
1.1 Functional role of assigned classes 3

2 List of issues **3**
2.1 Naming conventions 3
2.2 Indentation 4
2.3 Braces 4
2.4 File organization 4
2.5 Wrapping lines 4
2.6 Comments 5
2.7 Java Source Files 5
2.8 Class and interface declarations 5
2.9 Initialization and declarations 6
2.10 Method calls 6
2.11 Object comparison 6
2.12 Computation, comparisons and assignments 7
2.13 Exceptions 7

3 References **7**
3.1 Software and tools used 7
3.2 Hours of work 8

1 Assigned classes

We have been assigned only one method, that is `distributePrepare()` which belongs to the class `RegisteredResources`.

1.1 Functional role of assigned classes

The method `distributePrepare()` manages the decision to commit a transaction which involves multiple resources.

Every participant can vote to commit, read only or rollback the transaction, which will be committed only if every resource involved votes for the commit. If even a single participant vote for a rollback, an exception is thrown, then the application will need to go back through all the resources which have or may have already voted for a commit and try to roll them back, not including the one which raised the exception.

In the end, the method returns the overall vote to the caller, which will proceed to execute the actual commit or rollback.

2 List of issues

2.1 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.
  - `laoIndex`, `laoResource`: in the method it doesn't explain what does `lao` mean.
  - `rmErr`: it clearly represent some kind of error but it doesn't explain what the `rm` stands for.
- 3. Class names are nouns, in mixed case, with the first letter of each word in capitalized.
  - `INTERNAL` is all uppercase, which should be wrong but since it's not a custom class it should be fine like that (lines 550, 578).
- 5. Method names should be verbs, with the first letter of each addition word capitalized.
  - `commit_one_phase`: should be `commitOnePhase` (line 479)
  - `_release`: the name shouldn't start with the underscore (line 626)
- 7. Constants are declared using all uppercase with words separated by an underscore.

- `Completed`: it's a constant in the `ResourceStatus` class, so it should be all uppercase (lines 480, 490, 624).
- `Heuristic`: same as above (line 558).

2.2 Indentation

- 8. Three or four spaces are used for indentation and done so consistently.
  - Lines 505-506: only two spaces are used to indent.
  - Lines 525, 535-536, 582, 640-642, 650-652: should be indented one space less.
- 9. No tabs are used to indent.
  - Some lines (512, 514-516, 523-525, 531, 533-536, 591, 637, 639-642, 646, 648-651) are indented with tabs.

2.3 Braces

- 11. All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.
  - At lines 602 and 623 there are two single-statement `if`s that don't have curly braces.

2.4 File organization

- 13. Where practical, line length does not exceed 80 characters.
  - The only line that exceeds 80 characters is the 603 because it can't be split.

2.5 Wrapping lines

- 17. A new statement is aligned with the beginning of the expression at the same level as the previous line.
  - Lines 515-516, 585, 592: should be indented one tab less.
  - Lines 523-525, 533, 535-536, 637, 639, 646, 648: should be indented two tabs more.
  - Line 534, 640-642, 649-651: should be indented one tab more.

2.6 Comments

- 18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.
  - The ones in the method allow the user to understand how the method works but they could have written one too for the last `if` statement (line 635).
- 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.
  - Line 123-124: commented out variable declaration without any reference.
  - Line 126: this variable declaration has been commented out because it was moved to another class, as it's said in the comment.
  - Lines 498-499: the comments above explain what these lines are for but there's no clue about why they've been commented out. The fact that the variables declared are not used in the method probably means that these lines can be deleted.
  - Line 644: commented out method calling without any reference.

2.7 Java Source Files

The checks in this paragraph refer to the java file that contains the method that has been assigned to us.

- 23. Check that the javadoc is complete.
  - Line 661: javadoc is missing, but it's a getter method so it probably doesn't need it as it's already self-explanatory.

2.8 Class and interface declarations

- 25. Check that the class declarations are in the correct order.
  - Static variables aren't declared in the beginning, but mostly at the end.
  - The variables are all `private` except for `_logger`, which then should have been put first.

2.9 Initialization and declarations

- 30. Check that constructors are called when a new object is desired.
  - The objects `currResource` (line 466) and `checkProxy` (line 505) are created without calling the relative constructor.
- 31. Check that all object references are initialized before use.
  - At line 592 there is a used object without any reference.

2.10 Method calls

- 35. Check that the correct method is being called, or should it be a different method with a similar name.
  - Line 479: `commit_one_phase()` should be `commitOnePhase()`.
  - Line 505: `getProxyChecker()` seems that it doesn't exist. The callable methods on `Configuration` are:
    * `getConfiguration()`
    * `getInstance(...)`
    * `setConfiguration(...)`
  - Line 506: `isProxy(currResource)` doesn't correspond to any method in the class.
- 36. Check that method returned values are used properly.
  - The `set()` method called on an `ArrayList` replace the specified element by the index with the relative given by the parameter but returns the element previously at the specified position (lines 480, 490, 558, 624, 645, 655).

2.11 Object comparison

- 38. Check that all array (or other collection) indexes have been prevented from going out-of-bounds.
  - For the `resourceObjects` array there is only a getter with an index that returns the element at the specified position in this list. The for cycle exits when one of this condition is not verified:
    `i < nRes && result != Vote.VoteRollback;`
    Every time `nRes` was incremented, an item has been added to the array and that's why it isn't possible to go out of bounds.
    The same thing is valid for the `resourceStates` array too.
- 40. Check that all objects are compared with `.equals()` and not with "`==`".

- All the comparisons are done using "==" instead of `equals()`:
  * `laoResource == null` (line 469)
  * `result == Vote.VoteCommit` (line 470)
  * `currResult == null` (line 520)
  * `currResult == Vote.VoteCommit` (line 599)
  * `logSection == null` (line 602)
  * `result == Vote.VoteReadOnly` (line 608)
  * `currResult == Vote.VoteRollback` (line 629)
  * `result == Vote.VoteCommit` (line 635)

2.12 Computation, comparisons and assignments

- 46. Check the liberal use of parenthesis is used to avoid operator precedence problems.
  - Line 469: the highlighted parentheses in bold can be omitted because the precedence of the arithmetic and logic operators is preserved anyway: `(i == nRes - 1) && lastXAResCommit && (laoResource == null) &&`
- 51. Check that the code is free of any implicit type conversions.
  - `get(i)` returns the element at the specified position in this list. The program force this element to be a Resource type: `(Resource) resourceObjects.get(i);`

2.13 Exceptions

- 52. Check that the relevant exceptions are caught.
  - There is no throw-catch expression that ensure a no `IndexOutOfBoundsException` because the code already guarantee this case. But it's always better to have managed this exception in order to manage any error that can happen if, for example, something went wrong before.

3 References

3.1 Software and tools used

- LyX: to redact and format this document.
- Notepad++: to properly read the java file that has been assigned to us.
- TortoiseSVN: to download the code to inspect.

3.2 Hours of work

This is the time spent to redact this document:

- Massimo Schiavo: x hours.
- Marco Edoardo Cittar: x hours.