POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software Engineering 2 2016/2017
Apache OFBiz Project

Code Inspection

V 1.0

Redacted by:

Leonardo Chiappalupi
CP 10453536

Ivan Bugli
CP 10453746

Table of contents

1. Assigned classes

Application: apache-ofbiz-16.11.01

Package: org.apache.ofbiz.order.order

Class: OrderChangeHelper.java

2. Functional role of classes

The class is part of a sort of Apache OFBiz project, an open-source suite for businesses with various sub-applications for human resources management, marketing management, accounting, manufacturing, and every enterprise process automation. The class that has been assigned to us is part of this order management category, and it is a helper container of methods to perform changes for orders, such as:

- Approving orders;
- Rejecting orders;
- Canceling orders;
- Manage inventory reservations;
- Creating invoices;
- Holding orders;
- Manage and release payment authorizations.

This has been inferred by the package path and by the name and contents of the class, as well as by analyzing how other classes use the methods provided by this helper.

Additional information has been found reading the (indeed, not very detailed) documentation for the class available at the links in Section 5.3: *References*.

3. List of issues

The following list contains an extensive analysis on the code of the assigned class; for every element of the checklist (see Section 5.3: *References*), the source has been inspected and issues have been recorded. General notes are written in black, while actual issues are reported using the ⚠ icon on the left.

3.1 Naming Conventions

⚠ Lines 249, 282: not explicative variable name 'orh'

⚠ Line 250: not explicative variable name 'btparty'

⚠ Line 256: not explicative variable name 'opps'

⚠ Class attribute *module* is a constant, but does not uses constants naming conventions (not capitalized)

⚠ Method *orderStatusChanges* is probably not a verb (ambiguous)

3.2 Indention

⚠ Lines 89-93: bad indentation (should be 2 spaces more on the left)

⚠ Lines 127-131: bad indentation (should be 2 spaces more on the left)

3.3 Braces

All the document use KR-style braces

No if blocks have then instruction without braces

3.4 File Organization

⚠ Lines 18-19: there is no blank line separating license and import statements

⚠ Lines 39-40: there is no blank line separating comment block from class declaration

⚠ Many lines do not respect the usual 80 characters limit, namely:
Lines 46, 50, 51, 53, 69, 74, 77, 84, 85, 96, 97, 98, 101, 107, 109-111, 122, 123, 134-136, 139, 145, 147, 148, 150, 154, 158, 160, 164, 168, 170, 175, 177, 181, 185, 187, 192, 194, 197, 198, 200-203, 208, 210-214, 224, 226-229, 233-237, 241, 244, 261-266, 274, 277, 285, 286, 288, 294, 298.

⚠ Too many lines do not even respect the "mandatory" 120 characters limit, namely: Lines 50, 53, 74, 77, 96, 101, 111, 134, 139, 145, 147, 154, 187, 194, 200-203, 210-213, 224, 226, 233, 241, 274, 285.

3.5 Wrapping Lines

Line breaks are used regularly apart from line 262 which is broken at low level; however this line is so long that a correct line break is not possible.

Except for indentation issues found in section 3.2, all methods of the same (sub)level are aligned.

3.6 Comments

⚠ The class has only a few comments, perhaps too few detailed.
These comments are at lines: 56, 225.

There is no commented code.

3.7 Java Source Files

Public class is at the beginning of the file and it is the only class in the source file analyzed.

⚠ Javadoc is almost completely non-existant (except for very basic information about the class at lines 37-39).

3.8 Package and Imports

Package statements and import ones are arranged correctly, at the beginning of the source file.

3.9 Class and Interface Declarations

Class declaration order is respected, in relation to what is actually present in the source file.

Methods are grouped correctly (order approve/complete/cancel, order status, payments).

Methods *approveOrder* (line 50), *rejectOrder* (line 84), *cancelOrder* (line 122) are quite long (around 30 lines) but ok.

⚠ Methods *createReceivedPayments* (line 241) and, especially, *orderStatusChanges* (line 145) are definitely too long – and containing too nested expressions.

⚠ There is duplicated code: lines 95-104 and lines 133-142 are the same; lines 200-204 and 211-215 are the same, too.

⚠ Class is declared *final*, and this is likely a mistake; the class should be *static* instead.

3.10 Initialization and Declarations

Methods have right visibility (this is a helper class, all methods are public and all methods are to be available for other classes). The class itself, though, contains only static methods and variables, thus (being a helper) it should be declared as static.

⚠ Variables at lines:
57-59 (*HEADER_STATUS, ITEM_STATUS, DIGITAL_ITEM_STATUS*),
86-87 (*HEADER_STATUS, ITEM_STATUS*),
124-125 (*ORDER_CANCELLED, ITEM_CANCELLED*),
are actually constants, named with constant convention and used as constants, but they are not declared as final (static).

⚠ The above variables neither have the correct scope, since they are shared among different methods and should be constants of the whole class.

⚠ Variables at lines:
51 (*productStore*),
85 (*productStore*),
123 (*productStore*),
197 (*isDigital*),
200 (*digitalStatusFields*),
201 (*digitalStatusChange*),
148 (*statusResult*),
158 (*itemStatusResult*),
181 (*orderItemSeqId*),
147 (*statusFields*),
211 (*digitalStatusFields*),
154 (*itemStatusFields*),
208 (*orderItemType*),
212 (*digitalStatusChange*),
165 (*delegator*),
226 (*cancelInvFields*),
227 (*cancelInvResult*),
234 (*releaseFields*),
235 (*releaseResult*),
259 (*payments*),

262 (*results*),
250 (*btparty*),
249 (*orh*),
256 (*opps*),
283 (*items*),
286 (*serviceRes*),
282 (*orh*),
285 (*serviceParam*)
could be declared *final*.

The few new object initializations are correctly made using proper constructors.

All object references are initialized before used, or assigned with the try/catch construct.

All variables are initialized when declared, either directly or by return of other methods.

All declarations are made at the beginning of a method, when possible.

3.11 Method Calls

All methods seem to be called with the correct parameters, in the correct order.

All methods called seem to be the right ones.

No discrepancies between methods return values and objects they are assigned to.

⚠ Unnecessarily qualified static method calls (methods of this very class called with *OrderChangeHelper.*):
line 74 (*orderStatusChanges*);
line 75 (*releaseInitialOrderHold*);
line 96 (*orderStatusChanges*);
line 97 (*cancelInventoryReservations*);
line 98 (*releasePaymentAuthorizations*);
line 99 (*releaseInitialOrderHold*);
line 109 (*createReceivedPayments*);
line 110 (*createOrderInvoice*);
line 111 (*orderStatusChanged*);
line 134 (*orderStatusChanged*);
line 135 (*cancelInventoryReservations*);
line 136 (*releasePaymentAuthorizations*);
line 137 (*releaseInitialOrderHold*).

3.12 Arrays

No arrays are used in this class.

3.13 Object Comparison

Some comparisons (lines 52, 164, 198, 202, 209, 213, 228, 236, 258, 265) correctly use *equals*, including the ones about strings.

⚠ Many lines, though, contain the '==' (or '!=') operator; namely lines: 52, 62, 65, 68, 88, 91, 126, 129, 155, 164, 172, 189, 198, 248, 252, 281. However one should note that these are all comparisons with "null", so there is no practical difference between using *equals* and == (or !=).

3.14 Output Format

There are a few debug outputs, or error outputs injected into exceptions, and they have no grammatical errors or misspells.

⚠ Messages are very generic though: they only provide the order number the error is about, but no additional details on the nature of the error – with the exception of the error message for *IllegalArgumentException* at line 53. See outputs at lines 77, 101, 139, 229, 237 (they only refer to generic "problems", etc.).

3.15 Computation, Comparisons and Assignments

The class does not contain any *brutish programming* practice, nor useless redefinitions of built-in Java functions.

Order of computation is fine (no mathematical operators at all).

Booleans use is correct.

All try-catch blocks are legitimate, even if the exception raised is quite always the same generic one.

No implicit conversions of any sort.

3.16 Exceptions

⚠ Line 145: exception *GenericServiceException* is never thrown in the method

⚠ Line 224: exception *GenericServiceException* is never thrown in the method

⚠ Line 233: exception *GenericServiceException* is never thrown in the method

- ⚠ Line 241: exception *GenericEntityException* is never thrown in the method
- ⚠ Line 241: exception *GenericServiceException* is never thrown in the method
- ⚠ No catch block takes any sort of action except for logging the error.
- ⚠ Catch blocks at lines 112 and 114 have the same body, and can be combined.

3.17 Flow of Control

No switch statements to check.

Only 2 loops in the code, performed using *foreach* construct – correct.

3.18 Files

No files are used in the class whatsoever.

4. Additional problems

4.1 Hard-coded Strings

The class makes massive use of hardcoded strings, which is better not to use in an international context and to make the application future proof. Hardcoded strings are 89, spread across the code: while some may remain (e.g. logging errors ones), some are used as parameters or to initialize values and we suggest to implement another static final class to hold persistent parameters. Most of these strings are widely re-used across the code, for instance:

"orderId" (replicated 10 times), "orderItemSeqId" (replicated 3 times), "userLogin" (replicated 8 times), "orderHeader" (replicated 3 times), "statusId" (replicated 5 times), "changeOrderItemStatus" (replicated 3 times).

4.2 Redundant 'null' assignments

Several variables are initialized with 'null' values, but this assignment is redundant:

Line 166 (*orderHeader*),
Line 173 (*orderItems*),
Line 182 (*product*),
Line 190 (*productType*),
Line 242 (*orderHeader*),
Line 275 (*orderHeader*).

4.3 Cyclomatic complexity

The cyclomatic complexity of method *orderStatusChanges* (line 145) is 20, which is quite above the normal authorized threshold of 10.

4.4 Useless methods

Methods at lines 294 and 298 (*releaseInitialOrderHold* and *abortOrderProcessing*) simply return true, thus are useless and can be eliminated (or add a *TODO* statement if they are still to be implemented).

5. Appendices

5.1 Version history

V1.0 / 2017.02.05 / Initial Release

5.2 Hours of work

The total time spent per person redacting this paper is about: 10 hours.

5.3 References

POLITECNICO DI MILANO, **CODE INSPECTION ASSIGNMENT TASK DESCRIPTION.PDF**
For code inspection checklist

CAL POLY SAN LUIS OBISPO, **BRUTISH PROGRAMMING**
For checks on brute force programming techniques to be avoided

APACHE.ORG, **OFBIZ API TECHNICAL DOCUMENTATION**
https://ci.apache.org/projects/ofbiz/site/javadocs/index.html?help-doc.html

APACHE.ORG, **OFBIZ PROJECT DOCUMENTATION**
https://ofbiz.apache.org/documentation.html