Figure 1: Politecnico di Milano

Code Inspection Document

Version 1.0

Emanuele Ricciardelli (mat. 875221)

Giorgio Tavecchia (mat. 874716)

Francesco Vetro (mat. 877593)

February 5, 2017

Contents:

1. Code Description

1.1 Assigned classes

The classes assigned belong to the Apache OFBiz project and are the following:

- *WebPosEvents.java* located in the *org.apache.ofbiz.webpos* package;
- *WebPosSession.java* located in the *org.apache.ofbiz.webpos.session* package;

1.2 Functional role

1.2.1 WebPosSession

The *WebPosSession* class implements a customized type of session. It contains the whole informations about a session concerning a purchase(we don't know if it is associated to an e-commerce system or a shop-handling system) because it handles data about Product Store, Facility ID, Shopping Cart, Current UOM and all these are initialized when the class is created.
This class provides a set of functionalities that allow other classes to perform login, logout, parameter session settings and special login's based on a role (probably a role, different from the client, with respect to the system).
The process of Logout are managed through a special transaction, that is an instance of an external class classed *WebPosTransaction*.

1.2.2 WebPosEvents

The *WebPosEvents* class is composed by static methods built to serve three different kind of functionalities. Mainly, its methods are invoked when a new HttpRequest is performed, so when a purchase instance is started, and they provide all the functionalities to handle this kind of event.
First, there are methods with the purpose of establishing a stable correlation between the HttpSession and the *WebPosSession* when the POS performs login creating a valid *WebPosSession* based on various information, involving the user's one, reported at the previous point in order to map the HttpSession information into a suitable structure.

So this class performs all the functionalities to provide a suitable *WebPosSession* and to handle its instance also by checking whether it already exists or removing it when it becomes useless (generally when a sale is completed).
About this last possibility, the second kind of functionality performed by this class is to provide operations in order to complete a sale.
At last, under a proper HttpRequest, the class provides the set containing all the features linked with a specific item: for example, if an item is chosen, the method retrieves from an XML database all the features and all the variants available for that object (e.g. its colour, its measures and so on).

2. Code Issues

2.1 Notation

In the document, we will use the following notations, now exposed to clarify the meaning of each of them:

- Single lines of code are denoted as **L.123.**
- Multiple sequential lines of code are denoted with the interval **L.123-456.**
- Specific points in the code inspection checklist are denoted as **C.1**, **C.2**,... See the final chapter for the full list.

2.2 Code issues

In this chapter we will provide a list of all the the issues found in the assigned classes. Only element of the checklist that present any issue are figured out, this means that if a point isn't mentioned, the related check was successful.
We divide this chapter in two section, one for each class assigned.

2.2.1 WebPosSession Class

1. **C.1.** The following elements are defined with name that could be more meaningful or correct:
    - **L.43**, **L.78**, **L.196**, **L.199**, **L.223** “module” : the string contains the class name, but the assigned name is not self-explanatory and does not allow to understand how it will be used.
    - **L.46** “attributes”: it represents the elements associated with the pos websession, however the name, excluded from its implementative sphere does not allow a proper comprehension.
2. **C.7. L.43.** The constant attribute “module” doesn’t follow the naming convention for constants and need to be correct in uppercases.
3. **C.13** & **C.14.** Different lines of code exceed the limit length of 80 and 120 characters since they’re not correctly broken. Here it is a list of them with the related correction.
    - **L.59** is 241 characters long, since it is a method signature, it can be correct by putting each argument below one another and indent it to clearly separate it from the body.
    - **L.171** is 105 characters long, it could be split after *“getString(“userLoginId”), “*. This just exceed the 80 characters’ limit.
    - **L.175** is 110 characters long, since it is a method signature, it could be split as previously defined for **L.59.**
      This just exceed the 80 characters’ limit.
    - **L.179** is 123 characters long, since it is a method signature, it could be split as previously defined for **L.59.**
    - **L.182** is 121 characters long, it could be split after *“"WebPosUiLabels", “*.
    - **L.185** is 121 characters long, it could be split after *“”PartyUiLabels",”*.
    - **L.188** is 121 characters long, it could be split after *“”PartyUiLabels",”*.
    - **L.194** is 126 characters long, it could be split after *“username, ”*.
    - **L.208** is 129 characters long, it could be split after *“"WebPosUiLabels", “*.
    - **L.215** is 83 characters long, it could be split after *“* || *“*.
    - **L.221** is 114 characters long, it could be split after *“false, “*.
4. **C.18** There are comments only in the *checkLogin* method, these are useful to the understanding of how the method works, providing an easier reading of the code. The other methods, otherwise, are unprovided.

5. **C.20** the code includes two public classes: the first is related to the same source file, ie *WebPosSession*, the second one, at **L.253**, concerns the definition of an exception.
6. **C.23** the javadoc of the *WebPosSession* class is absent, both in the same code, both on the site regarding the documentation; in the documentation only method signatures are shown, generated exceptions and fields, but with no description or explanation.
7. **C.27** the encapsulation of the class is not ensured since accessor methods (getter and setter) expose pointers of interval variables, allowing the direct changes from the outside. A possible solution to the problem is to modify the getter in order to provide a copy of the attribute to the outside and modify the setter to achieve an higher level of granularity to the possible modifications, preventing possible inconsistencies: i.e. the setters related to lists, can be modified in order to accept only one element of the list per time, in this way the element itself can be checked and rejected if it isn't an appropriate value or might be able to affect negatively the behaviour of other methods based on the same list.
8. **C.33** the following declarations are not positioned at the beginning of a block:
    - **L.218**
    - **L.219**

    in particular, at **L.218** the variable is declared and assigned at the same time but, since the assignment is the result of the previous *If Clause*, only the declaration with value *null* must be moved to the beginning of the block, leaving the assignment as it is.
9. **C.40** all objects are correctly compared with *equals*. At **L.207** it is used *"=="* to compare an object to a *null* value and it is correct.
10. **C.42** at **L.199** the error message stored via *"Debug.logError"* cites "Throwable caught!" but he mentions neither the cause, nor how to remedy to it.

2.2.2 WebPosEvents Class

1. **C1.** The following elements are defined with name that could be more meaningful or correct:
    - **L.52**, **L.116**, **L.213**, **L.218**, **L.228** "module" : the string contains the class name, but the assigned name is not self-explanatory and does not allow to understand how it will be used.
    - **L.129** "*completeSale"* name method: the method not only execute what the name suggests, but it also removes its *webPosSession* user. This is not implicit

in the name or in the javadoc (that is absent). It would be advisable to add this information.

- **L.168** "*virtualVariantMethodEnum"* the name would suggest the use of enumerations, but the code then works only with strings.
- **L.182** "*variantTree*" the name suggests the use of trees for the management of product variants, however, it is implemented and managed as a map.
- **L.186 "***featureOrder"*, in order to maintain a constant definition of variable names, since all lists name end with "List" suffix, it would advisable to correct it with "*FeatureOrderList"*.

2. **C.13** & **C.14.** Different lines of code exceed the limit length of 80 and 120 characters since they're not correctly broken. Here it is a list of them with the related correction.
   - **L.113** is 176 characters long, it could be split after *"productStoreId, "*.
   - **L.129** is 122 characters long, it could be split before the *Throws* keyword.
   - **L.141** is 139 characters long, it could be split before the *Throws* keyword.
   - **L.147** is 206 characters long, it could be split after "*webPosSession.getDelegator(), ".*
   - **L.167** is 139 characters long, it could be split after "*&&"* operator.
   - **L.175** is 129 characters long, it could be split after "*"getProductFeatureSet", "*.
   - **L.181** is 160 characters long, it could be split after "*productId, "*.
   - **L.182** is 128 characters long, it could be split after "*="*.
   - **L.190** is 201 characters long, it could be split after "*="*.
   - **L.195** is 124 characters long, it could be split after "*featureKey, "*.
   - **L.204** is 127 characters long, it could be split after "=".
   - **L.223** is 132 characters long, it could be split after "*"featureLists", ".*
3. **C.18** the class is totally devoid of comments (excepts a short one at **L.57**), which makes very difficult understanding the methods, it is so necessary to continuously consult the javadoc documentation which is, at the same time, very poor of contents.
4. **C.23** the javadoc of the *WebPosEvents* class is absent, both in the same code, both on the site regarding the documentation; in the documentation only method signatures are shown, generated exceptions and fields, but with no description or explanation.
5. **C.27** *getProductType* **L.153-233** : the method results to be very long, it is composed by a series of conditional statements (*If-Else clauses*) nested. This kind of structure makes the readability of the code complex, since the code itself it is composed as a macro functionality. Instead, the method could be decomposed into a series of sub methods, allowing a greater elasticity even in case of further changes.

6. **C.33** the following declarations are not positioned at the beginning of a block:
    - **L.61-63**
    - **L.135**
    - **L.176**
    - **L.182**
    - **L.186**
    - **L.204**

    The declarations in these lines should be moved at the beginning of the block. Since all of them are composed by declaration and assignment at the same time, the instruction should be splitted in a declaration part and in an assignment part. only the declaration should be moved and assigned at *null*, while the assignment, which is related to code flows, should not be moved.
7. **C.56 L.188-197** the only cycle in the code is properly formed, however, since it is a simple loop that iterates over a list of strings, it could be advisable to prefer writing "*for (Object element: e)"* easier to understand.

2.2.3 Other issues concerning the classes

As previously mentioned in **C.23**, one of the most serious problems encountered at this stage of inspection code was found in the complete lack of documentation of the classes.
The understanding of the code is therefore particularly arduous, forcing us to inspect other classes to give a meaning as precise as possible to the studied methods; the lack of documentation (or a minimal amount of comments) is then translates into the possibility to extend errors of misunderstanding. The wrong understanding of a secondary class can lead to incorrect understanding of that assigned to us. All this bring into a significant slowdown and the possibility of misapplication of some of the checklist points.
Finally, it is important to remember how the Javadoc should be written as soon as possible after the drafting of the code, in order to be as detailed as possible and of course updated with each change.

Reference documents

- WebPosSession.java class;
- WebPosEvents.java class;
- Code Inspection Assignment Table.xlsx;
- Assignments AA 2016-2017.pdf;
- https://ci.apache.org/projects/ofbiz/site/javadocs for Javadoc related to previous classes;
- https://ofbiz.apache.org/ to read the scope and mission of the complete project.

Used tools

- Github: for version control
- GoogleDoc: to write the document
- Google Fogli: to manage the checklist

Hours of works

- Emanuele Ricciardelli: ~ 10 hours;
- Giorgio Tavecchia: ~ 10 hours;
- Francesco Vetrò: ~ 10 hours.