

***“Code Inspection Document”***

**Version 1.0** *(01/02/2017)*

Giorgio Marzorati (876546)

Aniel Rossi (877018)

Andrea Vaghi (877710)

INDEX OF CONTENTS

CLASSES 3

FUNCTIONAL ROLES 3

LIST OF ISSUES 4

OTHER PROBLEMS 4

**INTRODUCTION**

The Code Inspection process aims to analyse the source code of a set of classes extracted from an existing project (a release of Apache OFBiz in this case), with the purpose of finding mistakes overlooking during the development phase. It is a crucial part of the software life cycle that aims to find bugs and code issues that cannot be discovered directly with unit test; it is particularly suited in situation when there is lack of executability (for example during the design phase, or the early development one). This examination has ben performed systematically with the support of the provided checklist, after the review of some of the OFBiz related reference documents (API, Framework, etc.) that can be found on the website.

# **ASSIGNED CLASSES**

The cluster of classes that were assigned to our team is the following (the full path location is shown):

apache-ofbiz-16.11.01/framework/minilang/src/main/java/org/apache/ofbiz/minilang/method/envops/**OrderMapList.java**

apache-ofbiz-16.11.01/applications/product/src/main/java/org/apache/ofbiz/product/config/**ProductConfigWorker.java**

A description of the roles belonging to the above mentioned class is explained in the next paragraph.

# FUNCTIONAL ROLES

## Overall OFBiz description:

In this first section, we first provide a brief description about what Apache OFBiz is.

The open-source project essentially provides various modules for enterprise management solutions (including ERP, CRM, E-Commerce and more) in web applications.

**ProductConfigWrapper.java**

This class represents a component that was made ad hoc to handle the instances of the ProductConfigWrapper Class. A ProductConfigWrapper is an object associated to a product of a generic e-commerce platform, that contains derived informations about it depending from the request, for example the productStoreId, the currency, the catalogId, the productStoreId etc.

This class contains 4 methods that coincides with the main operations that are done on a ProductConfigWrapper instance. They are:

\_getProductConfigWrapper

\_fillProductConfigWrapper

\_storeProductConfigWrapper

\_loadProductConfigWrapper

1)The first method (getProductConfigWrapper) returns a ProductConfigWrapper depending from the request that is passed to it. In particular, in the first place it searches if there is already a ProductConfigWrapper cached for the values that are passed through the request. If there is, it returns that value without doing anything new, if there's not, the Worker creates another instance of ProductConfigWrapper with the read parameters.

After the creation, the worker puts in the cache the new ProductConfigWrapper.

2)The second method does some operations on a ProductConfigWrapper instance in reference to an HttpServletRequest object.

In the first place the Worker extracts some vectors of Strings from the request.

If this vector is empty the Worker searches for Strings named as comment among the request parameters and the ones that are present are tagged in the current ProductConfigWrapper as "selected".

If the vector is not empty the Worker gets the selected features from it and checks them.

3)The third method (storeProductConfigWrapper) first search persisted configurations, and then updates the configWrapper.configId value if found.

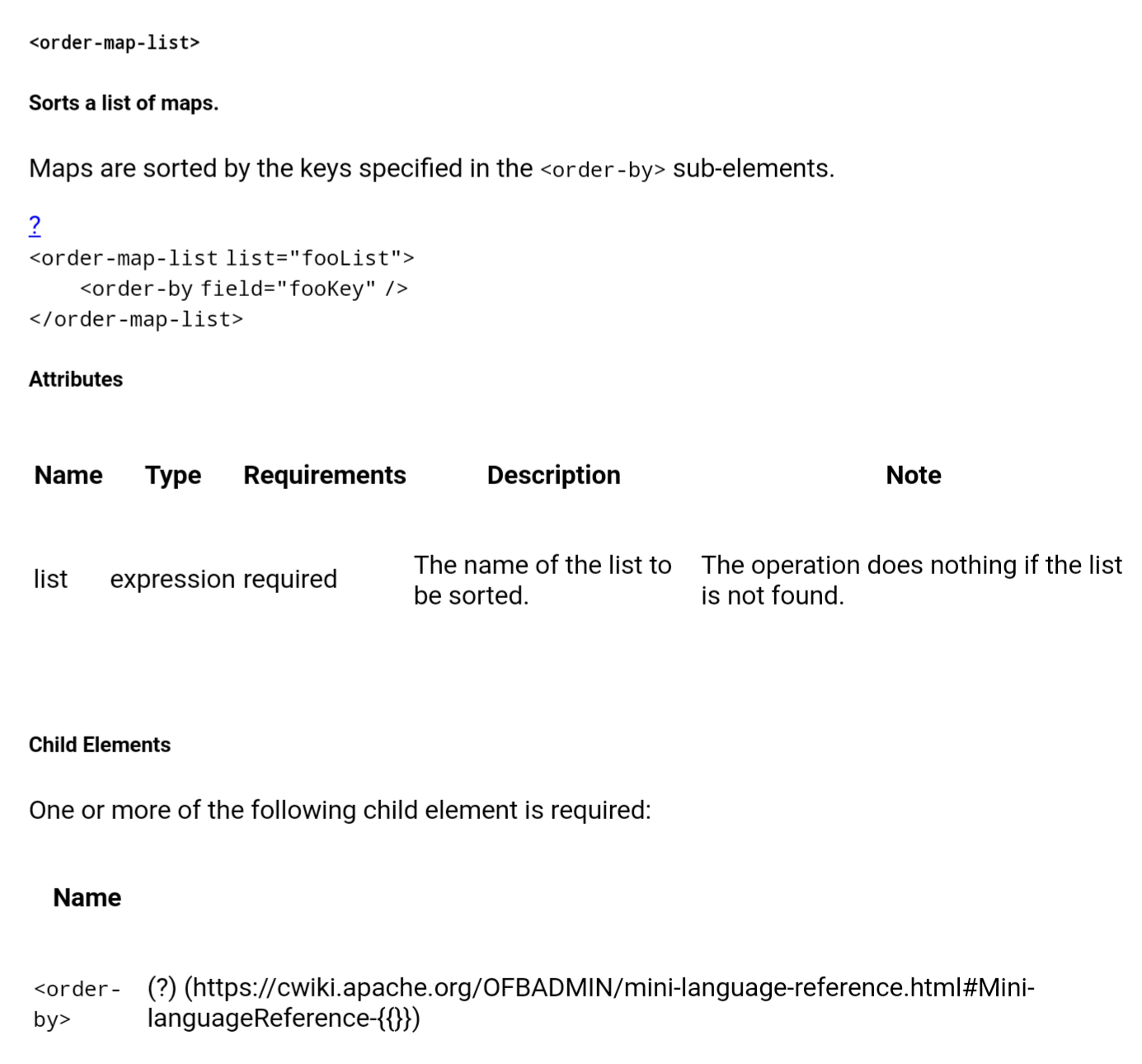
Otherwise it stores the ProductConfigWrapper to ProductConfigConfig entity and updates configWrapper.configId value with new configId.This method persists only the selected options, the price data is lost.

4)The fourth method (loadProductConfigWrapper) creates a new ProductConfigWrapper for the productId that is passed to it and configures it according to ProductConfigConfig entity with configId value.

ProductConfigConfig entity stores only the selected options, and the product price is calculated from input params.

**OrderMapList.java**

Apache OFBiz makes use of the Mini-language script engine, with which services and commands are defined in XML element, that are first parsed in a DOM tree and then into Java model objects. Services can be invoked directly from code. In our case, the class OrderMapList is the implementation of the <order-map-list> element, which is described in the Mini-language OZBiz reference document in this way:



Essentially, the OrderMapList is an utility class that allows a list of Map entities to be sorted according to specific keys included in the <order-by> sub-element.

# LIST OF ISSUES

OrderMapList.java

Naming Convention

|  |  |  |
| --- | --- | --- |
| Row | Code | Issue |
| 44 | FlexibleMapAccessor<List<Map<Object, Object>>> listFma | Name of the variable not meaningful |
| 49 | MiniLangValidate.validationOn() | Name of the method not a verb |
| 57 | UtilXml.childElementList(element, "order-by"); | Name of the method not a verb |
| 71 | public boolean exec(MethodContext methodContext) | Name of the method not a verb and not meaningful |
| 75 | listFma.get(methodContext.getEnvMap()) | Name of the method not meaningful |
| 84 | StringBuilder sb = new StringBuilder("<order-map-list "); | Name of the variable not meaningful |

File Organization

|  |  |
| --- | --- |
| Row | Issue |
| 59,61,94 | Line length exceeds 120 characters |

Output Format

|  |  |  |
| --- | --- | --- |
| Row | Code | Issue |
| 73 | throw new MiniLangRuntimeException("order-by sub-elements not found.", this); | Error message not self explaining doesn’t provide a guidance on how to correct the problem |

Java Source Files

|  |  |
| --- | --- |
| Row | Issue |
| 106 | Missing Javadoc for the overriden exec method |

ProductConfigWorker

Naming Convention

|  |  |  |
| --- | --- | --- |
| Row | Code | Issue |
| 52 | public static final String module = ProductConfigWorker.class.getName(); | Name of the variable not meaningful |
| 92 | String[] opts = request.getParameterValues(Integer.toString(k)); | Name of the variable not meaningful |
| 127 | configWrapper.getItemOtion(k, cnt); | Bad method spelling |

Indention

|  |  |
| --- | --- |
| Row | Issue |
| 244,252 | Odd indention |

Braces

|  |  |
| --- | --- |
| Row | Issue |
| 190 | Missing braces for single instruction if |

File Organization

|  |  |
| --- | --- |
| Row | Issue |
| 238,240,247,268,272,2278,290,313,351,393,396,198,215,221,160,165,144,77,69,57 | Line length exceeds 120 characters |
| 145 | Missing blank line |
| 227,231,294,304 | Useless blank line |

Class and Interface Declarations

|  |  |
| --- | --- |
| Row | Issue |
| 55 | Constructor should be placed before methods |

Output Format

|  |  |
| --- | --- |
| Row | Issue |
| 83,104,176,400 | Use of generic exceptions without the handling or the explaining of the specific problems that can arise |

Flow of control

|  |  |
| --- | --- |
| Row | Issue |
| 114,195,348,333,315,310 | Loop not correctly formed |

# OTHER PROBLEMS

In general not enough comments to make the cose understandable from an external inspector, and a poor documentation (javadoc). To understand the meaning of the classes we had to inspect in all the documentation of the application searching for almost every single class present in the classes assigned to us.