POLITECNICO
MILANO 1863

"Code Inspection Document"

Version 1.0 *(03/02/2017)*

Giorgio Marzorati (876546)
Aniel Rossi (877018)
Andrea Vaghi (877710)

INDEX OF CONTENTS

ASSIGNED CLASSES 3

FUNCTIONAL ROLES 4

Overall OFBiz description: 4

Roles description: 4

ProductConfigWrapper.java 4

OrderMapList.java 6

References: 6

LIST OF ISSUES 7

OrderMapList.java 7

ProductConfigWorker.java 8

OTHER PROBLEMS 9

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.
OFBiz is built on a framework that supports all the components provided.
The overall architecture can be divided in 3 layers:

- Presentation layer for the client side, which deals with the rendering of OFBiz pages;
- Business layer, which defines and implements services provided to the users
- Data layer, which is the component responsible for the data access. It is placed between the Business Layer and the database.

Roles description:

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 on the client 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 performed on a ProductConfigWrapper instance. They are:

- getProductConfigWrapper(String *productId*, String *currencyUomId*,
  HttpServletRequest *request*)

- fillProductConfigWrapper(ProductConfigWrapper *configWrapper*,
  HttpServletRequest *request*)

- storeProductConfigWrapper(ProductConfigWrapper *configWrapper*,
  Delegator *delegator*)

- loadProductConfigWrapper(Delegator *delegator*, LocalDispatcher *dispatcher*,
  String *configId*, String *productId*,
  String *productStoreId*, String *catalogId*,
  String *webSiteId*, String *currencyUomId*,
  Locale *locale*, GenericValue *autoUserLogin*)

1. The first method (getProductConfigWrapper) returns a ProductConfigWrapper depending from request data, which is passed as a parameter. 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 the value is not null, it returns that value, if it is, 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.

Every database operation in these methods, is performed through the Delegator interface, which provides database access methods for CRUD operations.

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 OFBiz reference document in this way:

`<order-map-list>`

Sorts a list of maps.

Maps are sorted by the keys specified in the `<order-by>` sub-elements.

```
?
<order-map-list list="fooList">
    <order-by field="fooKey" />
</order-map-list>
```

Attributes

Name	Type	Requirements	Description	Note
list	expression	required	The name of the list to be sorted.	The operation does nothing if the list is not found.

Child Elements

One or more of the following child element is required:

Name	
`<order-by>`	(?) (https://cwiki.apache.org/OFBADMIN/mini-language-reference.html#Mini-languageReference-{{}})

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.

References:

Due to the poor Javadoc of the assigned classes, in order to retrieve informations about their roles, we had to inspect most of the classes strictly related and included in the source code (both Javadoc and code analysis). Also, we derived usefull informations from entity diagrams on the website and framework documents (Minilang Refences etc.)

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 is not a verb
57	UtilXml.childElementList(element, "order-by");	Name of the method is not a verb
71	public boolean exec(MethodContext methodContext)	Name of the method is 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.java

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
238,240,247,268 272,278,290,313 351,393,396,198 215,221,160,165 144,77,69,57	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.