Code Inspection

Software Engineering 2
PowerEnJoy

Authors:

Bianchi Alessandro (Mat. 875035)
Canciani Francesco (Mat. 807056)
Cannone Lorenzo (Mat. 875802)

Document version: 1.0

Academic year:
2016-2017
Release date:
05-02-2017

Contents

1.Classes assigned to the group

Our team has only been assigned to a singular class of the entire Apache OFBiz project's release, located in the org.apache.ofbiz.product.category package, which is the following:

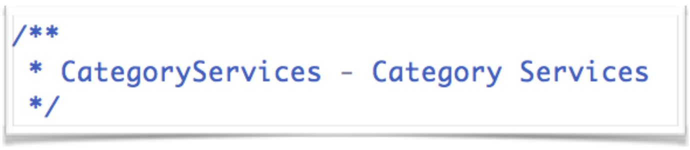
1.1 Revision History

Java class	Namespace pattern
CategoryServices.java	../apache-ofbiz-16.11.01/applications/product/src/main/java/org/apache/ofbiz/product/category/CategoryServices.java

2.Functional role of the assigned set of classes

This section provides an overview of the functional role of the CategoryServices class, in the general context of the whole Apache OFBiz project. Apache OFBiz is an open source product for the automation of enterprise processes that includes framework components and business applications for ERP, CRM and other business-oriented functionalities.

The overview is not helpful in understanding what the class' primary purpose is:



```
/**
 * CategoryServices - Category Services
 */
```

According to the explanation provided in OFBiz documentation, "services" are referred as:

"[...] independent pieces of logic which when placed together process many different types of business requirements. Services can be of many different types: Workflow, Rules, Java, SOAP, BeanShell, etc. A service with the type Java is much like an event where it is a static method, however with the Services Framework we do not limit to web based applications. Services require input parameters to be in a Map and the results are returned in a Map as well. This is nice since a Map can be serialized and stored or passed via HTTP (SOAP)." [source: https://cwiki.apache.org/confluence/display/OFBIZ/Service+Engine+Guide]

The previous definition clarifies the reason for which methods are declared as being static and why three of them return Maps and receive them as parameters. We can assume that this class is used within a combination of other services to accomplish larger tasks than that of the latter.

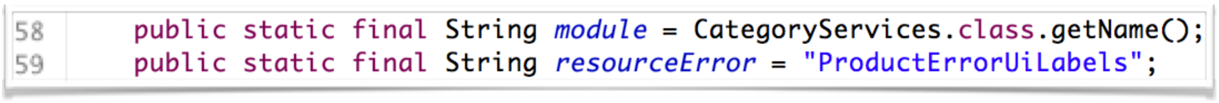
From the documentation, we also conceive that this class, like the other services class of the project, runs under the control of the Job Scheduler. Furthermore, by still going through the project's documentation, we get that products (for sale or for use) are organized into categories, and categories themselves are also organized into catalogs as well.

In conclusion, the primary purpose of CategoryServices appears to be the one of providing static utility methods to iterate and navigate through product categories.

3.List of issues found by applying the checklist

This chapter highlights all the bad practices that we have recognized during the inspection of the CategoryServices class. The issues we identified have been found by meticulously applying the checklist that was provided in the assignment document.

3.1 Naming convention



```
    public static final String module = CategoryServices.class.getName();
    public static final String resourceError = "ProductErrorUiLabels";
```

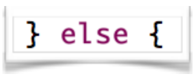
- The only two attributes of the class are constants. However, these two constants are not declared using all uppercase with words separated by underscores.

3.2 Indention

- No issues found. The indention all over the class is consistent and denoted by four spaces.

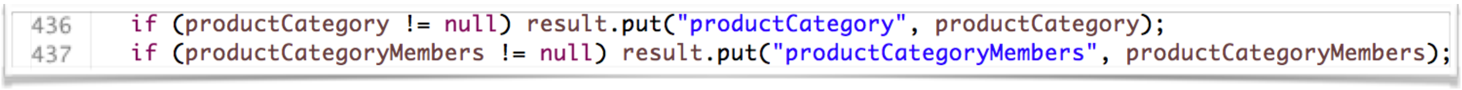
3.3 Braces

- All the braces are consistent with the "Kernighan and Ritchie" style. The "else" and the "catch" branches begin in line right after the closing braces belonging to the corresponding "if" and "try" blocks.



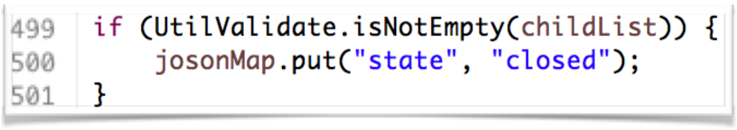
```
} else {
```

- There are some "if" statements, that have only one statement to execute, that are not surrounded by curly braces. For example:



```
436    if (productCategory != null) result.put("productCategory", productCategory);
437    if (productCategoryMembers != null) result.put("productCategoryMembers", productCategoryMembers);
```

This violation occurs in lines: 71, 99, 220, 436, 437.
However, some of the single-instruction "if" statements do have braces:



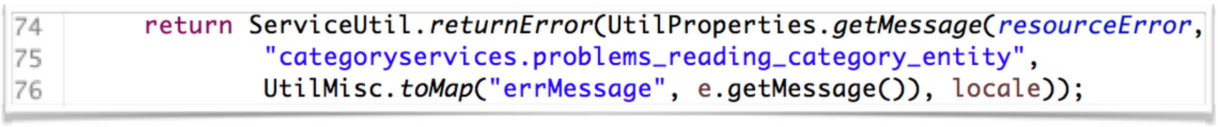
```
if (UtilValidate.isNotEmpty(childList)) {
    josonMap.put("state", "closed");
}
```

3.4 Final Organization

- Blank lines are used along the file, however, they are not only used to separate methods from one another. The developer uses one empty line as a separator both to divide methods from one another and inside a method itself. This approach does not improve the readability of the file given that it does not give a clear glance of the organization of the file.
- There are lot of lines which length exceeds 80 and 120 characters, with the highest peak in line 297 with 291 chars.

3.5 Wrapping Lines

- There are just two functional line breaks that avoid excessive line length of a return expression:

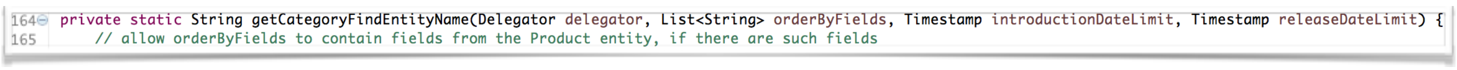


```
    return ServiceUtil.returnError(UtilProperties.getMessage(resourceError,
            "categoryservices.problems_reading_category_entity",
            UtilMisc.toMap("errMessage", e.getMessage()), locale));
```

The resulting extra statements are not aligned with the beginning of the original one.

3.6 Comments

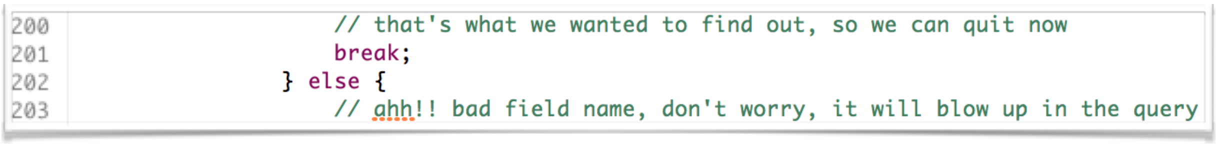
- In general this class lacks comments. There is only one method that comes with a meaningful explanation:



```
private static String getCategoryFindEntityName(Delegator delegator, List<String> orderByFields, Timestamp introductionDateLimit, Timestamp releaseDateLimit) {
    // allow orderByFields to contain fields from the Product entity, if there are such fields
```

  The other comments in the code describe their upcoming concerning instruction, such as expected behaviors, the choice of some data structure or variable names. We suppose that this comments lack comes from the fact that each method should be self explanatory thanks to its name.
  Nevertheless, the behavior of each method is not trivially intelligible because of their length, the amount of different variables and their reliance on other classes that are not introduced (with comments) in this file.

- Comments definitely need an improvement. Some of them are not really "business like" type of comments:



```
                        // that's what we wanted to find out, so we can quit now
                        break;
                    } else {
                        // ahh!! bad field name, don't worry, it will blow up in the query
```

3.7 Java Source Files

- In this file, the javadoc is completely missing. This does not help the user in understanding the purpose of the implemented methods and even the class itself.

3.8 Package and Import Statements

- No issues found.

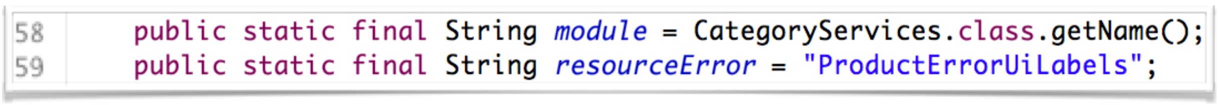
3.9 Class and Interface Declarations

- Every declaration is correct and follows the same order of the others.
- Since the five methods in the class are all utility ones (precisely getter methods) the order in which they are introduced is not crucial.

- Four out of five methods share the same order of length, which is a number of lines within one hundred lines of code.
  The remaining one instead (getProductCategoryAndLimitedMembers), is the biggest one with 239 lines of code. This method could be easily splitted into two or more dependent methods.

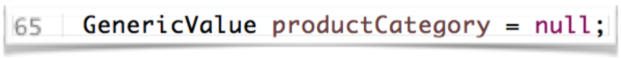
3.10 Initialization and Declarations

- The only two attributes in the file are final and static ones, but they are not used outside of this class, so they could be instantiated as private. Instead, they are declared as public attributes:

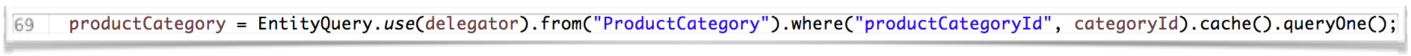


```
    public static final String module = CategoryServices.class.getName();
    public static final String resourceError = "ProductErrorUiLabels";
```

- This object is not created by calling its constructor:

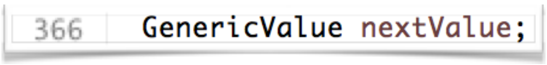


```
GenericValue productCategory = null;
```

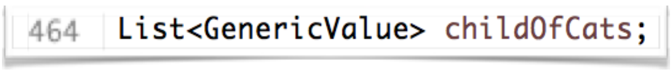


```
productCategory = EntityQuery.use(delegator).from("ProductCategory").where("productCategoryId", categoryId).cache().queryOne();
```

- These variables are not initialized when declared:

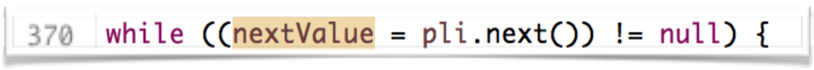


```
GenericValue nextValue;
```

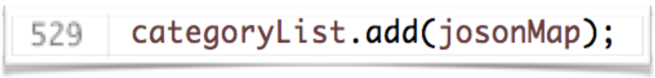


```
List<GenericValue> childOfCats;
```

They are used like this in the following lines:

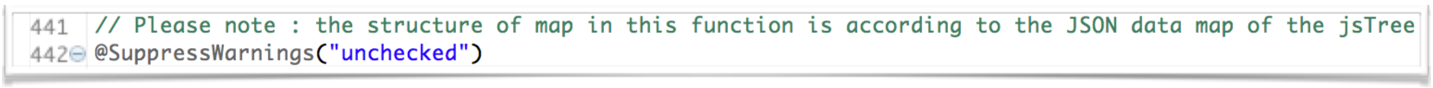


```
while ((nextValue = pli.next()) != null) {
```



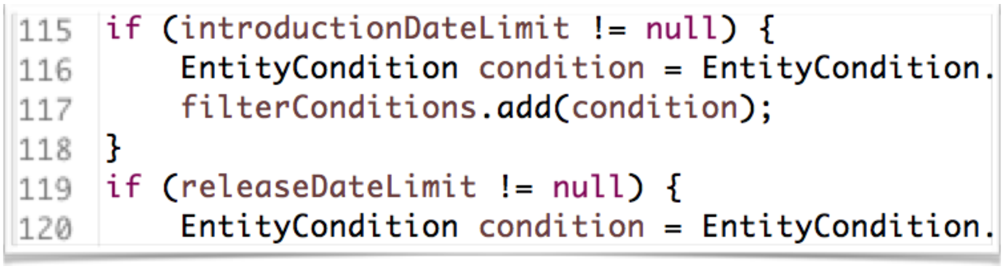
```
categoryList.add(josonMap);
```

Given the following comment that occurred in the file, we assume that the non-initialization of this type of variable has to be brought back to the rules of the format cited in the comment:



```
// Please note : the structure of map in this function is according to the JSON data map of the jsTree
@SuppressWarnings("unchecked")
```

- Sometimes variables are declared inside a block:



```
115 if (introductionDateLimit != null) {
116     EntityCondition condition = EntityCondition.
117     filterConditions.add(condition);
118 }
119 if (releaseDateLimit != null) {
120     EntityCondition condition = EntityCondition.
```

This behavior is present also in the following lines: 277, 297, 301, 343, 355, 366, 371, 483, 484, 489, 490, 496, 502, 503, 504, 505, 515, 522, 531.

3.11 Method Calls

- The method call EntityCondition.makeCondition(param1, param2, param3) occurs in lines 297 and 301. In both the occurrences of the cited method, the method receives as first and third parameter, a nested call to the same method. Those nested calls make the main method invocation hard to read (the entire line is not even splitted in pieces).
  The method is also present in line 355 but it only receives two parameters, so there must be an overloading version of it in the class EntityCondition, because of the above mentioned calls. Both methods return the same type of object: EntityCondition.

3.12 Arrays

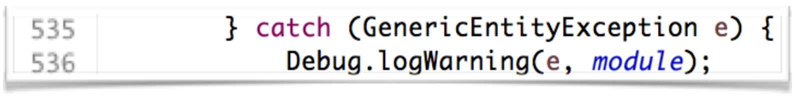
- Array declarations are not present in this class. The only other member of the Java Collections Framework present in the file is the linked list, which is properly managed.

3.13 Object Comparisons

- The equals method is used correctly.
  Despite that, there are some comparisons that are made adopting the "==" operator. However, these comparison only involve null values, so we figured that the the developer might not have wanted to throw NullPointerExceptions in these occurrences.

3.14 Output Format

- Most of the outputs are well formed and clear, however, some catch branches do not have comprehensive debug messages:

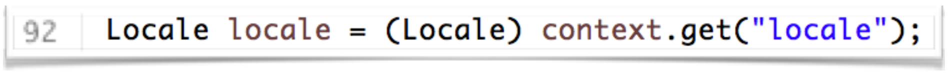


```
535          } catch (GenericEntityException e) {
536              Debug.logWarning(e, module);
```

This also happens in lines: 255, 282, 415, 423, 535.

3.15 Computation, Comparisons and Assignments

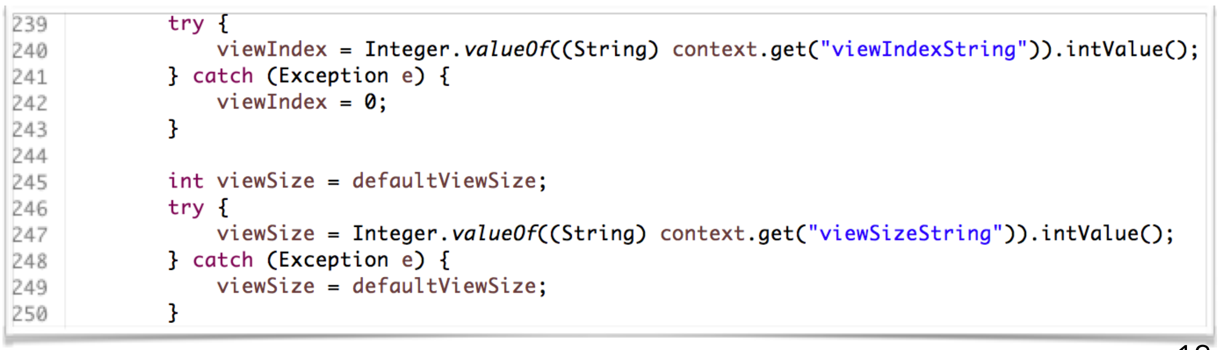
- The method getPreviousNextProducts(param1, param2) contains four “return” statements, which could be classified as “brutish programming”.
- Since this is an utility class used to iterate and navigate through items and their categories, there is not much computation to perform, especially for what concerns the mathematical perspective. The logic expressions seem to be correct with respect to the boolean comparisons.
- Any possible type conversion is preceded by correct casting:



```
92   Locale locale = (Locale) context.get("locale");
```

3.16 Exceptions

- Each method implemented in the class does not re-throw any exception.
  All of them just catch already thrown exceptions, some of which are specifically defined by the developer: GenericException and GenericEntityException.
- The overall number of catch branches is 11, but not all of them manage customized exceptions: catches at lines 241 and 248 handle generic exceptions, which might be justified by the implementation of the library method valueOf(String s) (for the Integer class) which throws NumberFormatException.

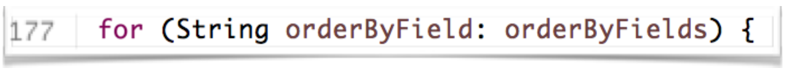


```
239         try {
240             viewIndex = Integer.valueOf((String) context.get("viewIndexString")).intValue();
241         } catch (Exception e) {
242             viewIndex = 0;
243         }
244
245         int viewSize = defaultViewSize;
246         try {
247             viewSize = Integer.valueOf((String) context.get("viewSizeString")).intValue();
248         } catch (Exception e) {
249             viewSize = defaultViewSize;
250         }
```

- Some catch branches are present just to log potential errors, without defining special instructions for those cases. This issue is also exposed in lines: 282, 312, 401, 415, 423.

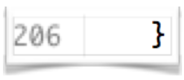
3.17 Flow of Control

- There is no switch statement in the file.
- There are just two cycles in this class and both of them are correctly formed:



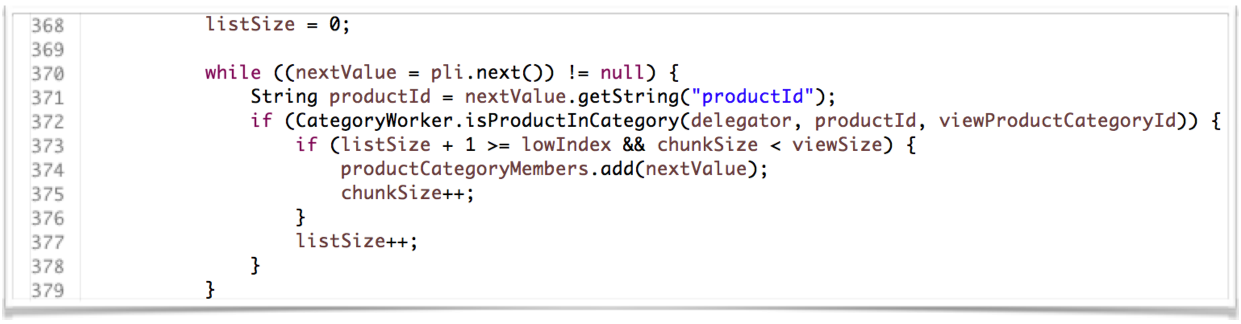
```
177    for (String orderByField: orderByFields) {
```

...



```
206        }
```

and



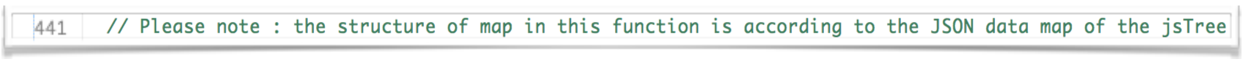
```
368             listSize = 0;
369 
370             while ((nextValue = pli.next()) != null) {
371                 String productId = nextValue.getString("productId");
372                 if (CategoryWorker.isProductInCategory(delegator, productId, viewProductCategoryId)) {
373                     if (listSize + 1 >= lowIndex && chunkSize < viewSize) {
374                         productCategoryMembers.add(nextValue);
375                         chunkSize++;
376                     }
377                     listSize++;
378                 }
379             }
```

3.18 Files

- Since there is no file opening or declaring, no issue about this topic can be found here.

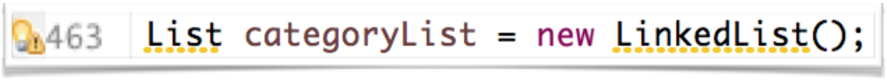
4.Other problems

- Some warnings have been found by Eclipse IDE in this class, but they are all limited within line 442 and line 540. Those lines concern the last method implemented in the file: getChildCategoryTree(HttpServletRequest request, HttpServletResponse response).
  All of these warnings refer to data structure declarations as Maps and Lists. However, this unusual kind of declaration is well foretold by the comment at line 441

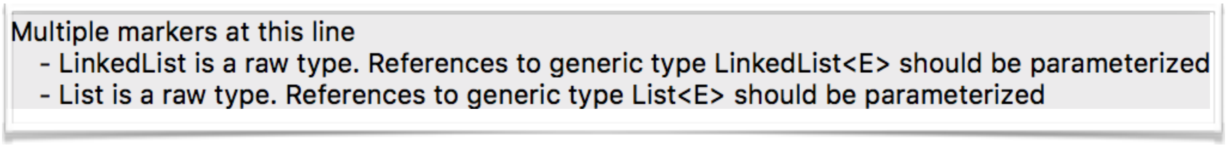


```
441   // Please note : the structure of map in this function is according to the JSON data map of the jsTree
```

  This comment warns about the structure of maps, although it does not mention linked lists, which are denoted by the following warnings:



```
List categoryList = new LinkedList();
```



Multiple markers at this line
- LinkedList is a raw type. References to generic type LinkedList<E> should be parameterized
- List is a raw type. References to generic type List<E> should be parameterized

  For this reason, this approach can be labeled as an inadvertence.

- In this class, there is a considerable amount of comparisons with "null" that do warn us that there might not be a proper management of null pointers inside the whole project which may affect the overall project stability.

5.Effort Spent

We spent approximately 7 hours each to compose this code inspection document.