POLITECNICO
MILANO 1863

Code Inspection Document

Guido Muscioni (mat. 876151)
Marco Orbelli (mat. 876649)
Paola Marchesini (mat. 876541)

Contents

1 Class assigned to the group

The class assigned comes from the Apache OFBiz project. That is an open source enterprise resource planning system. As all that kind of systems, it provides useful tools that simplify and automates business process, like the facilities management, the human resources management, etc.

The class assigned is called CatalogUrlFilter, is written in Java and it is contained in the product directory of the system code.

2 Functional role of assigned class

The role of the class assign to us is to filter the requests coming from the other system components. Due to the absence of any kind of documentation for the code given to us, our description of the functional role of the class can not be complete.

The class is used in order to create the correct link for the given request. Its flow is base on analyzing the categories and the product in the database at which the system refers. That analysis lead in the creation of an url that bring the applicant (that we do not know) both to the product, and the section, category, at which its belongs to.

The class is divided into 6 methods:

- *public void init(FilterConfig filterConfig) throws ServletException {...}*: this method simply configure the class;
- *public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {...}*: this is the main method of the class. Base on the analysis that we have made during the code ispection phase, its aim is to retrieve the productId and the categoryId for a specific;
- *public void destroy() {...}*: no info for that method;
- *public static String makeCategoryUrl(HttpServletRequest request, String previousCategoryId, String productCategoryId, String productId, String viewSize, String viewIndex, String viewSort, String searchString) {...}*: queries the DB in order to retrieve the product category;
- *public static String makeCategoryUrl(Delegator delegator, CategoryContentWrapper wrapper, List<String> trail, String contextPath, String previousCategoryId, String productCategoryId, String productId, String viewSize, String viewIndex, String viewSort, String searchString) {...}*: produce the real category url;
- *public static String makeProductUrl(HttpServletRequest request, String previousCategoryId, String productCategoryId, String productId) {...}*: queries the DB in order to retrieve the product;
- *public static String makeProductUrl(ProductContentWrapper wrapper, List<String> trail, String contextPath, String previousCategoryId, String productCategoryId, String productId) {...}*: produce the real product real;

3 List of issues found

3.1 Naming Conventions

1 All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.

- Line 235-236-237-238-259-260-261-262: use the variable "rolllupConds" while the correct name in english is rollup not rolllup.

2 If one-character variables are used, they are used only for temporary "throwaway" variables, such as those used in for loops.

- No issues about this checklist point have been found

3 Interface names should be capitalized like classes.

- No issues about this checklist point have been found

4 Method names should be verbs, with the first letter of each addition word capitalized. Examples: getBackground(); computeTemperature().

- No issues about this checklist point have been found

5 Class variables, also called attributes, are mixed case, but might begin with an underscore (' ') followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized. Examples: windowHeight,timeSeriesData.

- No issues about this checklist point have been found

6 Constants are declared using all uppercase with words separated by an underscore. Examples: MIN WIDTH; MAX HEIGHT.

- Line 52 ("module" should be written "MODULE") .

3.2 Indention

7 Three or four spaces are used for indentation and done so consistently.

- No issues about this checklist point have been found

8 No tabs are used to indent.

- No issues about this checklist point have been found

3.3 Braces

9 Consistent bracing style is used, either the preferred "Allman" style (first brace goes underneath the opening block) or the "Kernighan and Ritchie" style (first brace is on the same line of the instruction that opens the new block).

- The developers of that class use the style defined by "Kernigan and Ritchie" to manage the position of braces of the code. All the lines follow the phylosophy defined before, so no issues of that point were found.

10 All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.

- As it is mentioned before, all the code is coherent with the "Kernigan and Ritchie" style, so no line present that kind of issue.

3.4 File Organization

11 Blank lines and optional comments are used to separate sections (be- ginning comments, package/import statements, class/interface decla- rations which include class variable/attributes declarations, construc- tors, and methods).

- No issues about this checklist point have been found

12 Where practical, line length does not exceed 80 characters.

The lines that exceed 80 characters are:

- Line 68: 132 characters;
- Line 71: 113 characters;
- Line 74: 89 characters;
- Line 76: 112 characters;
- Line 77: 86 characters;
- Line 90: 98 characters;
- Line 91: 118 characters;
- Line 93: 190 characters;
- Line 95: 85 characters;
- Line 96: 92 characters;
- Line 97: 293 characters;
- Line 98: 91 characters;
- Line 99: 117 characters;
- Line 100: 135 characters;
- Line 102: 98 characters;
- Line 103: 95 characters;
- Line 104: 87 characters;
- Line 106: 102 characters;
- Line 107: 90 characters;
- Line 108: 113 characters;
- Line 109: 96 characters;
- Line 118: 234 characters;
- Line 119: 103 characters;
- Line 120: 128 characters;
- Line 121: 82 characters;
- Line 122: 98 characters;
- Line 124: 99 characters;
- Line 125: 91 characters;
- Line 127: 106 characters;
- Line 128: 94 characters;
- Line 129: 117 characters;
- Line 130: 100 characters;
- Line 145: 106 characters;
- Line 146: 126 characters;
- Line 147: 86 characters;
- Line 148: 214 characters;
- Line 150: 101 characters;
- Line 151: 100 characters;
- Line 152: 293 characters;
- Line 153: 91 characters;
- Line 154: 117 characters;
- Line 155: 135 characters;
- Line 157: 98 characters;
- Line 159: 99 characters;
- Line 160: 91 characters;
- Line 161: 81characters;
- Line 162: 112 characters;
- Line 163: 106 characters;
- Line 164: 141characters;
- Line 165: 114 characters;
- Line 166: 95 characters;
- Line 175: 234 characters;
- Line 176: 103 characters;
- Line 177: 128 characters;
- Line 179: 98 characters;
- Line 181: 99 characters;
- Line 182: 91 characters;

- Line 182: 91 characters;
- Line 183: 81 characters;
- Line 184: 112 characters;
- Line 185: 106 characters;
- Line 186: 141 characters;
- Line 187: 114 characters;
- Line 188: 95 characters;
- Line 201: 89 characters;
- Line 210: 84 characters;
- Line 211: 85 characters;
- Line 213: 207 characters;
- Line 215: 105 characters;
- Line 216: 97 characters;
- Line 219: 104 characters;
- Line 228: 91 characters;
- Line 228: 91 characters;
- Line 229: 155 characters;
- Line 230: 126 characters;
- Line 235: 91 characters;
- Line 236: 92 characters;
- Line 238: 186 characters;
- Line 239: 87 characters;
- Line 240: 102 characters;
- Line 247: 93 characters;
- Line 259: 95 characters;
- Line 260: 118 characters;
- Line 262: 190 characters;
- Line 264: 91 characters;
- Line 265: 95 characters;
- Line 266: 116 characters;
- Line 268: 81 characters;
- Line 277: 97 characters;
- Line 292: 97 characters;
- Line 295: 81 characters;
- Line 297: 98 characters;
- Line 300: 140 characters;
- Line 301: 84 characters;
- Line 307: 116 characters;
- Line 324: 140 characters;
- Line 325: 99 characters;
- Line 326: 97 characters;
- Line 332: 88 characters;
- Line 344: 214 characters;
- Line 347: 159 characters;
- Line 348: 98 characters;
- Line 350: 183 characters;
- Line 352: 98 characters;
- Line 357: 279 characters;
- Line 361: 108 characters;
- Line 364: 96 characters;
- Line 377: 99 characters;
- Line 384: 99 characters;
- Line 391: 99 characters;
- Line 398: 99 characters;
- Line 404: 92 characters;
- Line 412: 121 characters;
- Line 418: 140 characters;
- Line 422: 127 characters;
- Line 423: 88 characters;
- Line 425: 125 characters;
- Line 427: 89 characters;
- Line 433: 183 characters;
- Line 436: 108 characters;
- Line 439: 96 characters;
- Line 455: 121 characters;

13 When line length must exceed 80 characters, it does NOT exceed 120 characters.

The lines that exceed 120 characters are:

- Line 68: 132 characters;
- Line 93: 190 characters;
- Line 97: 293 characters;
- Line 100: 135 characters;
- Line 118: 234 characters;
- Line 120: 128 characters;
- Line 146: 126 characters;
- Line 148: 214 characters;
- Line 152: 293 characters;
- Line 155: 135 characters;
- Line 164: 141characters;
- Line 175: 234 characters;
- Line 177: 128 characters;
- Line 186: 141 characters;
- Line 213: 207 characters;
- Line 229: 155 characters;
- Line 230: 126 characters;
- Line 238: 186 characters;
- Line 262: 190 characters;
- Line 300: 140 characters;
- Line 324: 140 characters;
- Line 344: 214 characters;
- Line 347: 159 characters;
- Line 350: 183 characters;
- Line 357: 279 characters;
- Line 412: 121 characters;

- Line 418: 140 characters;
- Line 422: 127 characters;
- Line 425: 125 characters;
- Line 433: 183 characters;
- Line 455: 121 characters;

3.5 Wrapping Lines

14 Line break occurs after a comma or an operator.

- No issues about this checklist point have been found.

15 Higher-level breaks are used.

- No issues about this checklist point have been found.

16 A new statement is aligned with the beginning of the expression at the same level as the previous line.

- Line 297: shift of 4 space to the right to aligned the expression at the sam level of the previous line.
- To line 299 from 311: shift of 7 space to the right to put the expressions on the right level.

3.6 Comments

17 Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

The code is not explained in a good way. It lacks of comments to explain the geneal behaviour of the blocks of code which composed the class. In particular some comment must be added:

- Line 66 (to explain the general behaviour of "doFilter(..)");
- 117 (to explain in details the block of code);
- 174 (to explain in details the block of code);
- 418 and 433 (to explain the overload of the methods "makeProductUrl").

In addition the comments on the code lack of important details.

18 Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.

The comments on the code does not contain details about date and authors.

3.7 Java Source Files

19 Each Java source file contains a single public class or interface.

- No issues about this checklist point have been found

20 The public class is the first class or interface in the file.

- No issues about this checklist point have been found

21 Check that the external program interfaces are implemented consistently with what is described in the javadoc.

- No issues about this checklist point have been found

22 Check that the javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

- In our code there is no link to the javadoc

3.8 Package and Import Statements

23 If any package statements are needed, they should be the first noncomment statements. Import statements follow.

- No issues about this checklist point have been found.

3.9 Class and Interface Declarations

24 The class or interface declarations shall be in the following order.

- No issues about this checklist point have been found.

25 Methods are grouped by functionality rather than by scope or accessibility.

- No issues about this checklist point have been found.

26 Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

Duplicate Code

- There are two parts of the code which are duplicated: from line 156 to 167 and from line 178 to 189.

Long Methods

- doFilter(..): this method occurs 270 lines. It could be broken, designing new methods to do his sub-functionality.

Big Classes

- No issues about this checklist point have been found.

Breaking Encapsulation

- No issues about this checklist point have been found.

Coupling

- No issues about this checklist point have been found, we have only one class.

Cohesion

- No issues about this checklist point have been found.

3.10 Initialization and Declarations

27 Check that variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).

- Line 52: the variable "module" is declared "public final static String" while the correct way is "public static final String".

28 Check that variables are declared in the proper scope.

- Line 97: The variable "ContentAssocDataResourceViewTos" should be written with a lower case letter;
- Line 100: The variable "ElectronicText" should be written with a lower case letter;
- Line 120: The variable "ElectronicText" should be written with a lower case letter;
- Line 152: The variable "ContentAssocDataResourceViewTos" should be written with a lower case letter;
- Line 155: The variable "ElectronicText" should be written with a lower case letter;
- Line 177: The variable "ElectronicText" should be written with a lower case letter;

29 Check that constructors are called when a new object is desired.

- Line 90: In this constructor the type specification "<EntityCondition> " is redundant as the this type is implied;
- Line 145: In this constructor the type specification "<EntityCondition> " is redundant as the this type is implied;
- Line 210: In this constructor the type specification "<EntityCondition> " is redundant as the this type is implied;

- Line 253: In this constructor the type specification "<String>" is redundant as the this type is implied;
- Line 259: In this constructor the type specification "<EntityCondition> " is redundant as the this type is implied;
- Line 284: In this constructor the type specification "<String>" is redundant as the this type is implied;

30 Check that all object references are initialized before use.

- No issues about this checklist point have been found

31 Variables are initialized where they are declared, unless dependent upon a computation.

- No issues about this checklist point have been found

32 Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces '{' and '}'). The exception is a variable can be declared in a for loop.

- Line 105: In the declaration of "productIdStr" variable there is a useless assignment: "String productIdStr = null";
- Line 126 : In the declaration of "productIdStr" variable there is a useless assignment: "String productIdStr = null";
- Line 161: In the declaration of "productCategoryStr" variable there is a useless assignment: "String productCategoryStr = null";
- Line 183: In the declaration of "productCategoryStr" variable there is a useless assignment: "String productCategoryStr = null";
- Line 358: In the declaration of "url" variable there is a useless assigment: String url = "" ;
- Line 434: In the declaration of "url" variable there is a useless assigment: String url = "" ;

33 Other problems

- Line 357: the delegator is an unused method parameter;

3.11 Method Calls

34 Check that parameters are presented in the correct order

- No issues about this checklist point have been found.

35 Check that the correct method is being called, or should it be a different method with a similar name

- Line 289: at this line the method "size()" is invoked to evaluate the size of a list. It is better to invoke the method "isEmpty()".

36 Check that method returned values are used properly

- No issues about this checklist point have been found.

3.12 Arrays

37 Check that there are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).

- No issues about this checklist point have been found.

38 Check that all array (or other collection) indexes have been prevented from going out-of-bounds.

- No issues about this checklist point have been found.

39 Check that constructors are called when a new array item is desired.

- No issues about this checklist point have been found.

3.13 Output Format

40 Check that displayed output is free of spelling and grammatical errors.

- Line 352: the use of possessive case ("category's rule") is wrong. It cannot be used with object.
- Line 427: the use of possessive case ("product's rule") is wrong. It cannot be used with object.

41 Check that error messages are comprehensive and provide guidance as to how to correct the problem.

- Lines 247 and 277: the error messages are the same and it is impossible to distinguish the reason of this error and also how to correct the problem.

42 Check that the output is formatted correctly in terms of line stepping and spacing.

- No issues about this checklist point have been found.

3.14 Object Comparison

43 Check that all objects (including Strings) are compared with equals and not with ==.

- Line 101: if (ElectronicText != null);

- Line 156: if (ElectronicText != null);
- Line 178: if (ElectronicText != null);
- Line 283: if (trail == null);
- Line 294: if (trailElements.size() == 1);
- Line 296: if (trailElements.size() == 2);
- Line 316: if (productId != null);
- Line 364: if (urlBuilder.length() == 0 || urlBuilder.charAt(urlBuilder.length() - 1) != '/');
- Line 439: if (urlBuilder.length() == 0 || urlBuilder.charAt(urlBuilder.length() - 1) != '/');

3.15 Computation, Comparisons and Assignments

44 Check that the implementation avoids "brutish programming".

- Line 111, 132, 167, 189, 243, 270: in these lines, there are always "*break*" statements. Althought is not so difficult to understand why the breaks were put, usually a developer has to avoid the use of that kind of instructions. In actual fact, with that lines, the code becomes very unreadable;
- Line 101, 156, 178, 283, 316: these lines present always a conditional statement where null value is involved. That is a common brutish programming tendency, there is a function that return a binary value for nullity comparison. Moreover in the code UtilValidate class is use for empty comparison, so to standardize the code it could be used also in that lines;
- Line 371 to 373 and 446 to 448: the same function is used three times, but it can be used only once as it is shown in the JavaDoc of the append methods;
- All lines: as it is mentioned before the class is use to create url based on some requests. The building of the url, in that class, is performed by the StringBuilder class, however, java language contains itself a class used to build url: the URL class. Moreover there are open source codes that implement the URL class with more useful functionality (as UniRest and HttpComponents);
- Line 289: if (trail.size() > 0) it would better to use isEmpty() to check whether the collection is empty or not;

45 Check order of computation/evaluation, operator precedence and parenthesizing.

- Line 68 to 337: these are the lines occupied by the doFilter method. The order of computation of that method is not well defined: firstly it looks for the various ids, and after make something on them, but if the ids are not found in the correct way the method continues, always asking if the ids are "ok". That lead, if the ids are not found, in a waste of time. In actual fact there are many other ways to write that code to prevent this useless computation surplus;

46 Check the liberal use of parenthesis is used to avoid operator precedence problems.

- The code does not contains any numerical expression or something that need to take care about operator precedence.

47 Check that all denominators of a division are prevented from being zero.

- The code does not contains any kind of division, so no issue regarding that point have been found.

48 Check that integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.

- As it is mentioned in the two previous point no numerical or algebraic expressions is used, so it is not possible that issues regarding an unexpected result exist.

49 Check that the comparison and Boolean operators are correct.

- See the point 44 to read about the issues found regarding the comparison. However that point does not present issues, in actual fact the use of that kind of istructions is correct, but as it is mentioned in the reported point, it is not reccomended.

50 Check throw-catch expressions, and check that the error condition is actually legitimate.

- All the try-catch statement and error conditions are expressed in the correct way, so no issues about that have been found.

51 Check that the code is free of any implicit type conversions.

- No issues regarding that point have been found.

3.16 Exeptions

52 Check that the relevant exceptions are caught.

- Line 200, 218, 246, 276, 351, 426: in that lines a catch istruction is placed. All of them caught the correct exceptions, but it is not possible to understand what is the istructions that throws the exception. For example, the doFilter method contains two type of research one for the productId and one for the productCategoryId, but the exception is caught with the same statement so it is not possible to undestand what research fails.

53 Check that the appropriate action are taken for each catch block.

- As it is mentioned in the previous point, there is no way to understand what failure occur during the execution. Thus lead in unclear and very vague error message, and obviously the only action taken by the code is to log the exception with the generic error message.

3.17 Flow of Control

54 In a switch statement, check that all cases are addressed by break or return.

- No switch statement are presented, so no issues have been found for that point.

55 Check that all switch statements have a default branch.

- No switch statement are presented, so no issues have been found for that point.

56 Check that all loops are correctly formed, with the appropriate initialization, increment and termination expressions.

- Two types of loops are presented in the code, while and for statement. The first one is well formed. The second one is also well formed, all are expressed in the generalized form, but the termination expression is inside the code of the loop, written as a break statement. For more information go to point 44.

3.18 Files

57 Check that all files are properly declared and opened.

- No issues about this checklist point have been found.

58 Check that all files are closed properly, even in the case of an error.

- No issues about this checklist point have been found.

59 Check that EOF conditions are detected and handled correctly.

- No issues about this checklist point have been found.

60 Check that all file exceptions are caught and dealt with accordingly.

- No issues about this checklist point have been found.

4 Other problems detected

- Line 410: is better to introduce a new variable instead of reusing the parameter “trail”;
- Line 453: is better to introduce a new variable instead of reusing the parameter “trail”.

5 Appendices

5.1 Used Tools

- SonarQube: to make an initial analysis of the code;
- Lyx document processor: to write all the document;
- SourceTree: used as GitHub manager;
- GitHub: used to manage the shared building process of that document.

6 Hours of Work

Each member of the team spent 20 hours to build that document.