



POLITECNICO DI MILANO

SOFTWARE ENGINEERING 2 PROJECT WORK

---

---

# Code Inspection Assignment

---

Version: 1.1

*Authors:*

Jacopo FANTIN (mat. 878723)

Francesco LARGHI (mat. 876928)

*Professors:*

Elisabetta DI NITTO

Luca MOTTOLA

February 6, 2017



# Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Revision History . . . . .	1
1.2	Purpose and Scope . . . . .	1
1.3	Assigned class . . . . .	1
<b>2</b>	<b>Class Functional Role</b>	<b>3</b>
<b>3</b>	<b>List of Issues</b>	<b>5</b>
3.1	Naming Conventions . . . . .	5
3.2	Indention . . . . .	5
3.3	Braces . . . . .	5
3.4	File Organization . . . . .	6
3.5	Wrapping Lines . . . . .	6
3.6	Comments . . . . .	6
3.7	Java Source Files . . . . .	7
3.8	Package and Import Statements . . . . .	7
3.9	Class and Interface Declarations . . . . .	7
3.10	Initialization and Declarations . . . . .	8
<b>4</b>	<b>Appendix</b>	<b>9</b>
4.1	Used Tools . . . . .	9
4.2	Working Hours . . . . .	9



# 1. Introduction

## 1.1 Revision History

Version	Date	Author(s)	Summary
1.0	05/02/2017	Francesco Larghi - Jacopo Fantin	First release

## 1.2 Purpose and Scope

The purpose of this document is to point out any kind of error or mistake in the code of a java class, from syntax serious errors to simple style imperfections.

The document is written having as reference the *Code Inspection checklist* given to us and it's divided in different sections.

## 1.3 Assigned class

The class assigned to us is *PromoServices.java* in the package *org.apache.ofbiz.product.promo* that is part of **Apache OFBiz®**, an open source product for the automation of enterprise processes that includes framework components and business applications for ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), E-Business / E-Commerce, SCM (Supply Chain Management), MRP (Manufacturing Resource Planning), MMS/EAM (Maintenance Management System/Enterprise Asset Management).



## 2. Class Functional Role

This class is supposed to manage the promotion services as its name suggests. It creates, deletes and imports promotional codes. It has four methods with different tasks:

- *createProductPromoCodeSet:*

This method creates a set of new promotional codes taking all the informations needed from a passed argument called *context*. It has a main **for** cycle that generates a given "quantity" of random codes (of a given lenght) and checks the eventual existence of them in the database. If a code is unique it's accepted and appended in a new context inserted in the database. Finally, if no critical errors or exceptions are caught, it returns a success message.

- *purgeOldStoreAutoPromos:*

This method deletes all the old promotional codes not more available. As the first method, it takes all the informations from a given *context* and then starts to iterate a set of promotional codes that matches certain rules from the database and delete them. Finally, it returns a success message if no exceptions are caught.

- *importPromoCodesFromFile:*

Here the name is quite explicative. The method simply imports a set of promotional codes from a given File and validate them.

- *importPromoCodeEmailsFromFile:*

Also here the name is quite explicative. This method imports a set of emails from a given File and associate them with a promotional code.





## 3. List of Issues

### 3.1 Naming Conventions

The name of the class is *PromoServices* and it's quite explicative about the scope of the class: manage the loading, creation and deletion of promotion codes.

The method names are meaningful too in our opinion, they all do what they are supposed to do from their names:

- *createProductPromoCodeSet*
- *purgeOldStoreAutoPromos*
- *importPromoCodesFromFile*
- *importPromoCodeEmailsFromFile*

For what concerns the variables, of course there are many names, which are not always enough in order to entirely understand the complete meaning. Sometimes you have to check some lines of code to better understand the meaning of a variable.

Anyway there are not wrong or meaningless names, maybe just something too shortened, for example: *dctx* instead of *dispatchContext*.

There are not any other kind of naming errors or imperfections.

### 3.2 Indention

The indentation of the code uses four spaces and it's correct and consistent. There are not any kind of style errors.

### 3.3 Braces

The bracing style is always the "Kernighan and Ritchie" one, with the first brace on the same line of the instruction. Braces are never omitted for statements that have only one statement to execute.

### 3.4 File Organization

For what concerns the file organization we have to say that it's not perfect. The code lines frequently exceed 80 characters and sometimes more than 120 characters too, without any particular reason.

For example in the line 59 there is a list of all the available characters for a particular type of promotion code included in a big array and they reach 154 characters before going in a new line:

```
protected final static char[] smartChars = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
      '2', '3', '4', '5', '6', '7', '8', '9' };
```

We think that it's too much, they should divide the code in 3 lines instead of just 2. Anyway, there are many other examples like this, so we think that in general it's used a different convention in the line maximum length: 160 characters instead of 120.

### 3.5 Wrapping Lines

There are no errors in wrapping lines and all the statements are aligned correctly.

### 3.6 Comments

This is the biggest issue in the code: it's not well commented.

In particular the first method has only two comments and the second one is uncommented at all as you can see:

```
public static Map<String, Object> purgeOldStoreAutoPromos(DispatchContext dctx, Map<String, ? extends Object> context) {
    Delegator delegator = dctx.getDelegator();
    String productStoreId = (String) context.get("productStoreId");
    Locale locale = (Locale) context.get("locale");
    Timestamp nowTimestamp = UtilDateTime.nowTimestamp();

    List<EntityCondition> condList = new LinkedList<EntityCondition>();
    if (UtilValidate.isEmpty(productStoreId)) {
        condList.add(EntityCondition.makeCondition("productStoreId", EntityOperator.EQUALS, productStoreId));
    }
    condList.add(EntityCondition.makeCondition("userEntered", EntityOperator.EQUALS, "Y"));
    condList.add(EntityCondition.makeCondition("thruDate", EntityOperator.NOT_EQUAL, null));
    condList.add(EntityCondition.makeCondition("thruDate", EntityOperator.LESS_THAN, nowTimestamp));

    try {
        EntityListIterator eli = EntityQuery.use(delegator).from("ProductStorePromoAndAppl").where(condList).queryIterator();
        GenericValue productStorePromoAndAppl = null;
        while ((productStorePromoAndAppl = eli.next()) != null) {
            GenericValue productStorePromo = delegator.makeValue("ProductStorePromoAndAppl");
            productStorePromo.setAllFields(productStorePromoAndAppl, true, null, null);
            productStorePromo.remove();
        }
        eli.close();
    } catch (GenericEntityException e) {
        Debug.logError(e, "Error removing expired ProductStorePromo records: " + e.toString(), module);
        return ServiceUtil.returnError(UtilProperties.getMessage(resource,
            "ProductPromoCodeCannotBeRemoved", UtilMisc.toMap("errorString", e.toString(), locale)));
    }

    return ServiceUtil.returnSuccess();
}
```

So it's not so easy to understand what the class, the single method or the single statement is actually doing. The naming is correct, but the lack of comments implicates a lot of doubts and makes the code more difficult to read.

## 3.7 Java Source Files

The files *PromoServices.java* contains only the public class *PromoServices* so it's correct and consistent. The Javadoc is consistent too.

## 3.8 Package and Import Statements

There is only one package included placed as first statement after the initial comment. Then there are all the import statements as we expected.

## 3.9 Class and Interface Declarations

As we said before, the whole code is not well commented. In fact the class and the methods too are not commented at all. Anyway, the order of declarations is respected: firstly the class statement, then the public static variables and protected ones (there are not private static variables) and finally the methods (there are not the constructors). Methods are well ordered and free of duplicates, even if not commented.

## 3.10 Initialization and Declarations



## 4. Appendix

### 4.1 Used Tools

- TeXMaker: to create this pdf document
- Eclipse: to read and inspect the java code

### 4.2 Working Hours

Last Name	First Name	Total Hours
Larghi	Francesco	6h
Fantin	Jacopo	6h