Politecnico di Milano  
Industrial and Information Engineering  
Computer Science and Engineering

Software Engineering II Assignment

PowerEnJoy - car sharing

Academic Year: 2016/2017

Prof. Elisabetta Di Nitto

Alice Segato matr. 875045

Mark Edward Ferrer matr. 876650

Davide Bonacina matr. 876199

PowerEnJoy



Car Sharing App

Table of Content

[1 Classes Assigned 4](#_Toc473722266)

[2 Functional Role 4](#_Toc473722267)

[3 List of Issues 5](#_Toc473722268)

[3.1 Naming Conventions 5](#_Toc473722269)

[3.2 Indention 5](#_Toc473722270)

[3.3 Braces 5](#_Toc473722271)

[3.4 File Organization 6](#_Toc473722272)

[3.5 Wrapping Lines 6](#_Toc473722273)

[3.6 Comments 6](#_Toc473722274)

[3.7 Java Source Files 6](#_Toc473722275)

[3.8 Package and Import Statement 7](#_Toc473722276)

[3.9 Class and Interface Declarations 8](#_Toc473722277)

[3.10 Initialization and Declarations 8](#_Toc473722278)

[3.11 Method Calls 10](#_Toc473722279)

[3.12 Arrays 10](#_Toc473722280)

# 1 Classes Assigned

The class assigned to our group is CoreEvents.java that resides in the package org.apache.ofbiz.webapp.event of the OFBiz opensource application made by Apache and downloadable at <http://mirror.nohup.it/apache/ofbiz/apache-ofbiz-16.11.01.zip>.

# 2 Functional Role

Apache OFBiz is a product built for the automation of business process and includes framework components and business applications for ERP, CRM and many other business platforms. All of Apache OFBiz functionality are built upon a common framework. The functionality can be divided into the following distinct layers:

**Presentation Layer**

Apache OFBiz uses the concept of "screens" to represent the Apache OFBiz pages. Each page is, normally, represented as a screen. A page in Apache OFBiz consists of components. A component can be a header, footer, etc. When the page is rendered all the components are combined together as specified in the screen definition. Components might be Java Server Pages, FTL pages built around [FreeMarker](https://en.wikipedia.org/wiki/FreeMarker) [template engine](https://en.wikipedia.org/wiki/Template_engine_(web)), Forms and Menus Widgets. Widgets are an OFBiz specific technology.

**Business Layer**

The business, or application layer defines services provided to the user. The services can be of several types: Java methods, SOAP, simple services, workflow, etc. A service engine is responsible for invocation, transactions and security. Apache OFBiz uses a set of well established, open source technologies and standards such as [Java](https://en.wikipedia.org/wiki/Java_platform), [Java EE](https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition), [XML](https://en.wikipedia.org/wiki/XML) and [SOAP](https://en.wikipedia.org/wiki/SOAP). Although Apache OFBiz is built around the concepts used by Java EE, many of its concepts are implemented in different ways; either because Apache OFBiz was designed prior to many recent improvements in Java EE or because Apache OFBiz authors didn’t agree with those implementations.

**Data Layer**

The data layer is responsible for database access, storage and providing a common data interface to the Business layer. Data is accessed not in [Object Oriented](https://en.wikipedia.org/wiki/Object_Oriented) fashion but in a [relational](https://en.wikipedia.org/wiki/Relational_database) way. Each [entity](https://en.wikipedia.org/wiki/Entity) (represented as a row in the database) is provided to the business layer as a set of generic values. A generic value is not typed, so fields of an entity are accessed by the [column](https://en.wikipedia.org/wiki/Column_(database)) name.

CoreEvents is part of the package org.apache.ofbiz.webapp.event, which controls the event triggered from the web application, and dispatches these events to other components, probably linked to the presentation layer, in order to return a feedback to the user of the result of the operation.

Since the class contains only static methods, this means that it is a sort of utility class, which method are called by other components to perform actions on the services.

Since the Javadoc was not clear and complete, the exact role of this class cannot be deduced but we can assume that it takes an important place in the application in handling the events that come from the webapp related to specific framework components.

In particular there are three methods that return the result of a service, return Error(), returnNull(), returnSuccess().

the method getObjectFormServicePath() maps the results of previously run services with the path specified by the caller.

The method scheduleService() schedules a service for a specific time or recurrence of Request Parameters which are used for this service. Firstly checks that the requested service is available externally and then it maps it with the information inserted by the user through request parameters. This map will be used also to sync the services with the database in a second moment (this can be deduces from the comments in the method that recalls something about a sync with the database).

The method runService() runs a service specified by the caller: it retrieves the name of the service to invoke and the mode to set, checking if these information are available externally and then instantiates a new Event and calls its invoke() method to fire the service. If the invocation fails for any problem of the EventHandler, it fires an exception that notifies the caller of the error.

The method saveServiceResultToSession() is quite self explanatory: it saves the result of a service in the user’s session in order to allow the user to retrieve the result in any moment after the login in the webapp.

The method streamFile() retrieves the file pointed by the path that resides in the request parameter and streams its length in the browser calling the method UtilHttp.streamContentToBrowser().

# 3 List of Issues

## 3.1 Naming Conventions

* Variable seh does not represent clearly what it does at line 407;



* Constant variables module and err\_resource (respectively at line 60 and 61) are not in full upper case so they do not respect naming convention.



## 3.2 Indention

There are no issues according to this entry of the checklist.

## 3.3 Braces

* If statement at line 177, 179, 275 and 345 have no curly braces surrounding body;







## 3.4 File Organization

The lines that exceed the cap of 120 characters are: 149 (122), 160 (133), 165 (123), 214 (124), 228 (129), 233 (133), 245 (129), 258 (129), 266 (126), 282 (134), 318 (142), 321 (130), 338 (128), 340 (122), 345 (140), 373 (137), 360 (132), 425 (127), 444 (133), 449 (123), 455 (124) and 468 (132);

## 3.5 Wrapping Lines

Since there quite a bit lines that exceed 120 characters, there are no lines that have line breaks after a comma or an operator. The lines are indented in the correct way and consistently throughout the entire code of the CoreEvents.java class. Higher-level breaks are not necessary because there are not long operations (the lines are mostly method calls and/or assignments).

## 3.6 Comments

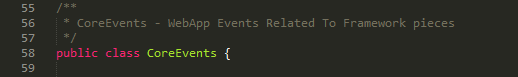
* Not all the code has comments explaining what a certain statement does;
* There are some instructions commented out at lines 110, 475 and 477 that can be removed.





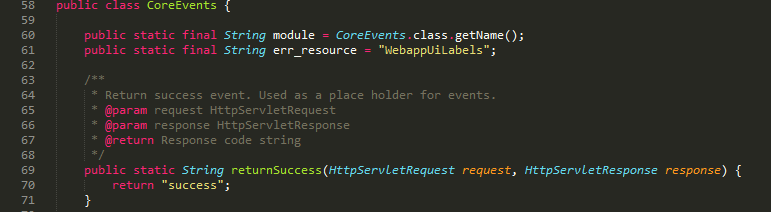
## 3.7 Java Source Files

The java file contains only one public class CoreEvents and it is the first listed in the file.

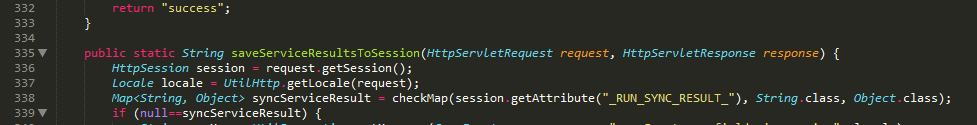


Some elements of the code of CoreEvents.java are not defined in the javadoc:

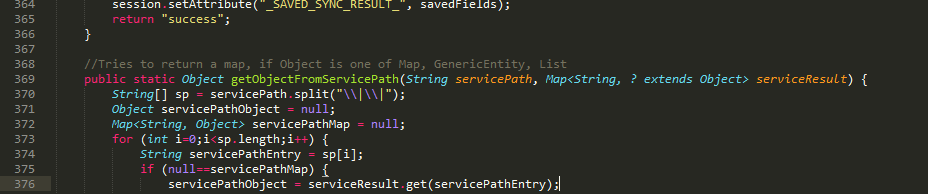
* public static final String module (line 60);
* public static final String err\_resource (line 61);



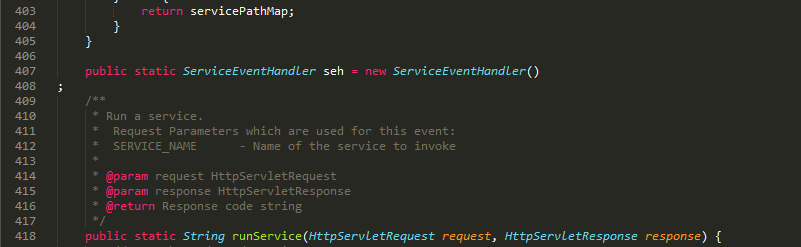
* public static String saveServiceResultsToSession() (line 335);



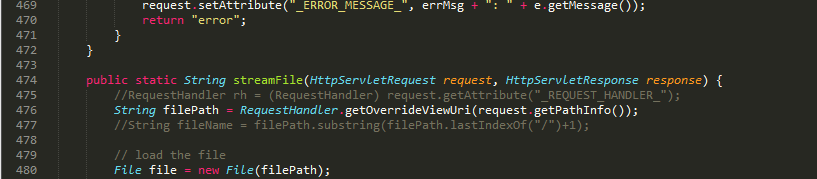
* public static Object getObjectFromServicePath() (line 369);



* public static ServiceEventHandler seh (line 407);



* public static String streamFile() (line 474);



## 3.8 Package and Import Statement

There are not problems regarding the import statements and the packages needed by the class. The only imports that can generate ambiguity are the static imports of checkCollection() and checkMap() methods from org.apache.ofbiz.base.util.UtilGenerics:



In any case, these static import cannot cause ambiguity because they import overloaded static methods and for this reason they differ for the list of parameters that each method get in input.

## 3.9 Class and Interface Declarations

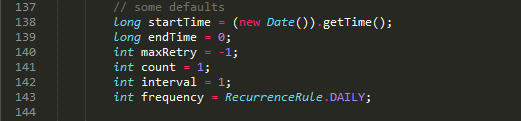
* There are three class static variables but one, public static ServiceEventHandler seh, is written in the middle of the methods in line 407. It should be written in line 62;
* The method public static String runService() should be written after the method public static String scheduleService() in order to group together the methods with linked functionalities.
* The method public static String scheduleService() is to long: it should be divided into sub methods in order to increase reusability and maintainability of the class itself.

## 3.10 Initialization and Declarations

The most used types in this class are the wrapper classes for the primitive types. For this reason, it could be better to declare all the variables that are currently in a primitive type, in their wrapper type, exploiting also the autoboxing and autounboxing of the variables introduced in Java 1.5. autoboxing and autounboxing allows to wrap primitive variables in their wrapper class in order to execute some operations that handles Object types (for example, declaring a Map with Long values cannot be done with long primitive type variables).

The variables that can be declared differently are:

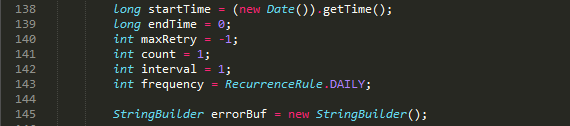
* long startTime = (new Date()).getTime(); [line 138]
* long endTime = 0; [line 139]
* int maxRetry = -1; [line 140]
* int count = 1; [line 141]
* int interval = 1; [line 142]
* int frequency = RecurrenceRule.DAILY; [line 143]



Can be declared as Long instead of long and Integer instead of int.

In the method public static StringScheduleService() the instance variables:

* long startTime = (new Date()).getTime(); [line 138]
* long endTime = 0; [line 139]
* int maxRetry = -1; [line 140]
* int count = 1; [line 141]
* int interval = 1; [line 142]
* int frequency = RecurrenceRule.DAILY; [line 143]
* StringBuilder errorBuf = new StringBuilder(); [line 145]
* ModelService modelService = null; [line 155]
* Map<String, Object> serviceContext = new HashMap<String, Object>(); [line 171]
* Map<String, Object> syncServiceResult = null; [line 312]









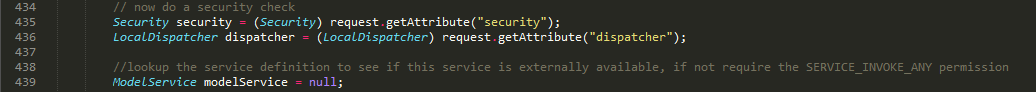
And in the method public static String saveServiceResultsToSession():

* Map<String, String[]> serviceFieldsToSave = checkMap(request.getParameterMap(), String.class, String[].class); [line 348]
* Map<String, Object> savedFields = new HashMap<String, Object>(); [line 349]



In the method public static String runService():

* Security security = (Security) request.getAttribute("security"); [line 435]
* LocalDispatcher dispatcher = (LocalDispatcher) request.getAttribute("dispatcher"); [line 436]
* ModelService modelService = null; [line 439]



These instance variable declarations should be written at the top of the block of each method in order to make the code more readable. The other variables that are not in this list must be declared in different positions because they need some additional information that cannot be retrieved at the beginning of the method or they are needed in control statements such as while, for and if.

In the declaration of the Map variables, the repetition of the value types in the constructor is redundant. The redundancies are in:

* Map<String, Object> serviceContext = new HashMap<String, Object>(); [line 171]



* Map<String, Object> savedFields = new HashMap<String, Object>(); [line 349]



* servicePathMap = new HashMap<String, Object>(); [line 386]



* servicePathMap = new HashMap<String, Object>(); [line 393]



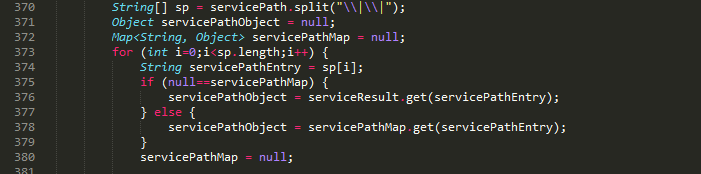
It is sufficient to write new HashMap<>() instead of specifying the types, that are already specified in the declaration statement.

## 3.11 Method Calls

There are no problems concerning the method calls.

## 3.12 Arrays

Since the only arrays used in this class are arrays of Strings, and since they are obtained by the use of the method String.split() (which never returns a null String[]), they have always at least one element and for this reason there is no problems with the for loop at line 373.



In the case of the other for loops, they are for-each loops so the iteration is safe due to the utilization of the iterators behind the scenes. The same situation occurs in the while loop at line 173 that iterates the elements of a map using an iterator for visiting the map.

