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

List of Figures

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 template engine, 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, Java EE, XML and 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 fashion but in a relational way. Each 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 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;

```
407     public static ServiceEventHandler seh = new ServiceEventHandler();
```

Figure 1

- 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.

```
60      public static final String module = CoreEvents.class.getName();
61      public static final String err_resource = "WebappUiLabels";
```

Figure 2

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;

```
177             if ("userLogin".equals(name)) continue;
178             // don't include locale, that is also taken care of below
179             if ("locale".equals(name)) continue;
```

```
275                 if (parsedValue > 0 && parsedValue < 8)
276                     frequency = parsedValue;
```

```
345     if (null!=request.getParameter("_CLEAR_PREVIOUS_PARAMS_") && request.getParameter("_CLEAR_PREVIOUS_PARAMS_").equalsIgnoreCase("on"))
346         session.removeAttribute("_SAVED_SYNC_RESULT_");
```

Figures 3, 4 and 5

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.

```
110         //Delegator delegator = (Delegator) request.getAttribute("delegator");
```

```
475         //RequestHandler rh = (RequestHandler) request.getAttribute("_REQUEST_HANDLER_");
476         String filePath = RequestHandler.getOverrideViewUri(request.getPathInfo());
477         //String fileName = filePath.substring(filePath.lastIndexOf("/")+1);
```

Figures 6 and 7

3.7 Java Source Files

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

```
55  /**
56   * CoreEvents - WebApp Events Related To Framework pieces
57   */
58  public class CoreEvents {
59
```

Figure 8

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);

```
58  public class CoreEvents {
59
60      public static final String module = CoreEvents.class.getName();
61      public static final String err_resource = "WebappUiLabels";
62
63      /**
64       * Return success event. Used as a place holder for events.
65       * @param request HttpServletRequest
66       * @param response HttpServletResponse
67       * @return Response code string
68       */
69      public static String returnSuccess(HttpServletRequest request, HttpServletResponse response) {
70          return "success";
71      }
```

Figure 9

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

```
332         return "success";
333     }
334
335     public static String saveServiceResultsToSession(HttpServletRequest request, HttpServletResponse response) {
336         HttpSession session = request.getSession();
337         Locale locale = UtilHttp.getLocale(request);
338         Map<String, Object> syncServiceResult = checkMap(session.getAttribute("_RUN_SYNC_RESULT_"), String.class, Object.class);
339         if (null==syncServiceResult) {
```

Figure 10

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

```
364         session.setAttribute("_SAVED_SYNC_RESULT_", savedFields);
365         return "success";
366     }
367
368     //Tries to return a map, if Object is one of Map, GenericEntity, List
369     public static Object getObjectFromServicePath(String servicePath, Map<String, ? extends Object> serviceResult) {
370         String[] sp = servicePath.split("\\|\\|");
371         Object servicePathObject = null;
372         Map<String, Object> servicePathMap = null;
373         for (int i=0;i<sp.length;i++) {
374             String servicePathEntry = sp[i];
375             if (null==servicePathMap) {
376                 servicePathObject = serviceResult.get(servicePathEntry);
```

Figure 11

- public static ServiceEventHandler seh (line 407);

```
403             return servicePathMap;
404         }
405     }
406
407     public static ServiceEventHandler seh = new ServiceEventHandler()
408 ;
409     /**
410      * Run a service.
411      *  Request Parameters which are used for this event:
412      *  SERVICE_NAME      - Name of the service to invoke
413      *
414      * @param request HttpServletRequest
415      * @param response HttpServletResponse
416      * @return Response code string
417      */
418     public static String runService(HttpServletRequest request, HttpServletResponse response) {
```

Figure 12

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

```
469             request.setAttribute("_ERROR_MESSAGE_", errMsg + ": " + e.getMessage());
470             return "error";
471         }
472     }
473
474     public static String streamFile(HttpServletRequest request, HttpServletResponse response) {
475         //RequestHandler rh = (RequestHandler) request.getAttribute("_REQUEST_HANDLER_");
476         String filePath = RequestHandler.getOverrideViewUri(request.getPathInfo());
477         //String fileName = filePath.substring(filePath.lastIndexOf("/")+1);
478
479         // load the file
480         File file = new File(filePath);
```

Figure 13

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:

```
20
21  import static org.apache.ofbiz.base.util.UtilGenerics.checkCollection;
22  import static org.apache.ofbiz.base.util.UtilGenerics.checkMap;
23
```

Figure 14

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]

```
137         // some defaults
138         long startTime = (new Date()).getTime();
139         long endTime = 0;
140         int maxRetry = -1;
141         int count = 1;
142         int interval = 1;
143         int frequency = RecurrenceRule.DAILY;
144
```

Figure 15

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]

```
138         long startTime = (new Date()).getTime();
139         long endTime = 0;
140         int maxRetry = -1;
141         int count = 1;
142         int interval = 1;
143         int frequency = RecurrenceRule.DAILY;
144
145         StringBuilder errorBuf = new StringBuilder();
```

```
155           ModelService modelService = null;
```

```
171       Map<String, Object> serviceContext = new HashMap<String, Object>();
```

```
312         Map<String, Object> syncServiceResult = null;
```

Figures 16, 17, 18 and 19

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]

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

Figure 20

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]

```
434         // now do a security check
435         Security security = (Security) request.getAttribute("security");
436         LocalDispatcher dispatcher = (LocalDispatcher) request.getAttribute("dispatcher");
437
438         //lookup the service definition to see if this service is externally available, if not require the SERVICE_INVOKE_ANY permission
439         ModelService modelService = null;
```

Figure 21

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]

```
171         Map<String, Object> serviceContext = new HashMap<String, Object>();
```

Figure 22

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

```
349         Map<String, Object> savedFields = new HashMap<String, Object>();
```

Figure 23

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

```
386                 servicePathMap = new HashMap<String, Object>();
```

Figure 24

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

```
393                 servicePathMap = new HashMap<String, Object>();
```

Figure 25

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.

```
370         String[] sp = servicePath.split("\\|\\|");
371         Object servicePathObject = null;
372         Map<String, Object> servicePathMap = null;
373         for (int i=0;i<sp.length;i++) {
374             String servicePathEntry = sp[i];
375             if (null==servicePathMap) {
376                 servicePathObject = serviceResult.get(servicePathEntry);
377             } else {
378                 servicePathObject = servicePathMap.get(servicePathEntry);
379             }
380             servicePathMap = null;
```

Figure 26

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.

```
170         // make the context valid; using the makeValid method from ModelService
171         Map<String, Object> serviceContext = new HashMap<String, Object>();
172         Iterator<String> ci = modelService.getInParamNames().iterator();
173         while (ci.hasNext()) {
174             String name = ci.next();
175
```

Figure 27

3.13 Object comparison

All the == used are for checking if objects are null or a value is 0 in all the other cases the method equals() is used.

3.14 Output Format

The output is free from spelling errors. The only spelling errors that occur are due to the unconventional use of multiple words appended to each other without the upper case on the second one. These occur only in error message strings at lines 61 (Webapp instead of WebApp),

```
ource = "WebappUiLabels";
```

Figure 28

160 and 444 (modelservice instead of modelService)

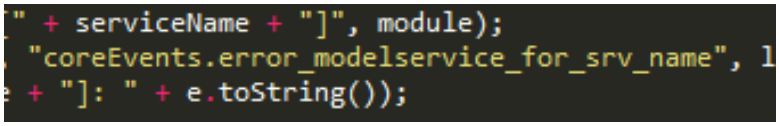

Figure 29

and 468 (eventhandler instead of eventHandler).

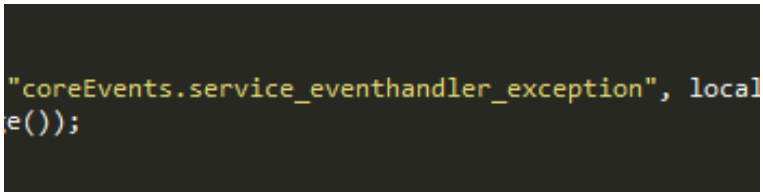

Figure 30

Every error message returns "error" but also reports in the console useful debug information. Even if all errors are handled, it is impossible to troubleshoot the type of error in case of multiple error logs from other components of the application. This makes harder to solve possible problems that occur at run time.

The output is correctly formatted in terms of line stepping and spaces.

The only double spaces are in the code comments after the full stop at line 3, 5, 8 and 15 and at lines 97, 98, 99 and 100 probably due to alignment problems but this does not break the code itself.

3.15 Computation, Comparison and Assignment

"Brutish programming" is avoided.

There are no expressions that require to check the order of computation or evaluation in this class.

There are no expressions in the code that need liberal use of parenthesis to avoid operator precedence problem since there aren't operations that make use of the mathematical operators

No divisions are made.

No integer arithmetic error.

All the comparisons and boolean operators are correct.

Throw-catch conditions and error conditions are legitimate.

Implicit conversion at line 139. The declaration of a long variable to 0 should be 0L.

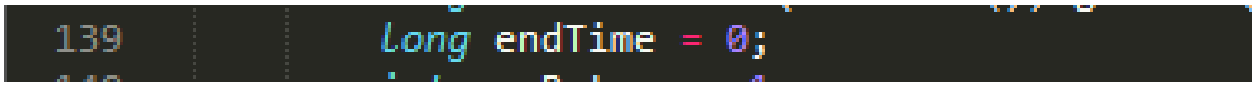

Figure 31

3.16 Exceptions

All relevant exceptions are caught: even if they are all handled, they always return a error string so it decreases the possibility to fix the problem when an exception rises. They should at least report the type of exception and what raised that error.

Not all the catch blocks handle correctly the errors that rise from the computation: some of them just return an error string.

3.17 Flow of Control

There are no switch statement in this class.

Every loop is safe because they make use of iterators to iterate through the maps and the arrays of Strings.

3.18 Files

The files are properly declared and opened.

The files are not properly closed if an error occurs at line 484-487. A solution to avoid this could be declaring the variable fis out of the try statement and to add fis.close() in the catch statements.

There are no EOF to detect.

There are no exceptions not handled.

```
480             File file = new File(filePath);
481             if (file.exists()) {
482                 Long longLen = Long.valueOf(file.length());
483                 int length = longLen.intValue();
484                 try {
485                     FileInputStream fis = new FileInputStream(file);
486                     UtilHttp.streamContentToBrowser(response, fis, length, null);
487                     fis.close();
488                 } catch (FileNotFoundException e) {
489                     Debug.logError(e, module);
490                     return "error";
491                 } catch (IOException e) {
492                     Debug.logError(e, module);
493                     return "error";
494                 }
495             }
496             return null;
497         }
498     }
```

Figure 32

4 Effort Spent

ALICE				
GIORNO	ORA INIZIO	ORA FINE	OGGETTO	ORE LAVORO
31/01	14.00	17.00	Code inspection	3.00.00
01/01	14.00.00	20.00.00	Code inspection	6.00.00
02/01	17.30.00	20.30.00	Code inspection	3.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				TOTALE
				12.00.00

DAVIDE				
GIORNO	ORA INIZIO	ORA FINE	OGGETTO	ORE LAVORO
30/1	14.30	17.30	Code inspection	3.00.00
31/01	11.00.00	16.00.00	Code inspection	5.00.00
03/01	18.00	20.00.00	Code inspection	2.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				TOTALE
				10.00.00

MARK				
GIORNO	ORA INIZIO	ORA FINE	OGGETTO	ORE LAVORO
30/1/2017	10.00	17.00.00	Code inspection	7.00.00
31/01	11.00.00	16.00.00	Code inspection	5.00.00
31/1/17	20.00.00	22.00.00	Code Inspection	2.00.00
01/02	10.30	14.00.00	Started writing Code inspection Document	3.30.00
03/02	17.20.00	18.30.00	Final layouting of the Code Inspection Document	1.10.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				0.00.00
				TOTALE
				18.40.00