Politecnico di Milano

A.A. 2016–2017

Software Engineering 2: "PowerEnJoy"

Code Inspection Document

Pietro Ferretti, Nicole Gervasoni, Danilo Labanca

February 5, 2017

Contents

1 Assigned Class

We were assigned one class to analyse, located in:

```
apache-ofbiz-16.11.01/framework/service/
    src/main/java/org/apache/ofbiz/service/job/JobManager.java
```

2 Functional Role

We found some information about the class we've been assigned and the environment it belongs to on Apache OFBiz's website[1] and its wiki[2].

2.1 Apache OFBiz

Apache OfBiz is an open source ERP system for the automation of enterprise processes. It offers many different applications and components to integrate and manage business processes like resource management, activities planning and customer relationship management.

2.2 Service Engine Framework

The Service Engine is one of the components of Apache OFBiz.

Services are 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.

Services are defined through the Service Definition and are assigned to a specific Service Engine. Each Service Engine is responsible for invoking the defined service in an appropriate way. Services can be scheduled to run at specific times to run in the background via the Job Scheduler.

2.3 Job Scheduler

The Job Scheduler (also called Job Manager) is integrated with the services framework, and is tasked with accepting new jobs to schedule and running them at the correct time.

The scheduler is a multi-threaded component with a single thread used for job managing/scheduling and separate threads used for invocation of each service. When a job is scheduled to run, the scheduler will call the service dispatcher associated with the job to invoke the service in its own thread. This will prevent long or time consuming jobs from slowing down other jobs in the queue.

---

[1] https://ofbiz.apache.org/
[2] https://cwiki.apache.org/confluence/display/OFBIZ/Home

2.4 Assigned Class: Job Manager

We can find more specific details about the class in the class Javadoc:

```
1
2  /**
3   * Job manager. The job manager queues and manages jobs. Client
4   * code can queue a job to be run immediately by calling the
5   * runJob({@link #runJob(Job)}) method, or schedule a job to be
6   * run later by calling the {@link #schedule(String, String,
7   * String, Map, long, int, int, int, long, int)} method.
8   * Scheduled jobs are persisted in the JobSandbox entity.
9   *
10  * <p>A scheduled job's start time is an approximation - the
11  * actual start time will depend on the job manager/job poller
12  * configuration (poll interval) and the load on the server.
13  * Scheduled jobs might be rescheduled if the server is busy.
14  * Therefore, applications requiring a precise job start time
15  * should use a different mechanism to schedule the job.</p>
16  */
```

The Javadoc describes two public methods: `runJob` and `schedule`, respectively used to execute a job immediately (or at least as soon as the resources are available), and to schedule a job to be run at a later time.

There are two relevant methods in the class that are not mentioned in the main Javadoc:

- `reloadCrashedJobs` is a public method that checks the status of all scheduled jobs in the queue, and reschedules every job that has crashed, if there are any;

- `poll` is a protected method that checks the job queue and returns a list of all the jobs that are scheduled to run.

3 List of Issues

We list here all the issues we have found in the code all the issues corresponding to each point in the checklist

3.1 Naming Conventions

1. Meaningful Names   Ok. All names are meaningful.

2. One-character variables   Ok, there are no one-character variables.

3. Class names   Ok, every class name is in mixed case and properly capitalized.

4. Interface names OK. No interfaces are declared. (se ce ne sono) every interface used by the code is in mixed case and properly capitalized.

5. Method names Ok. Every method name is a verb and every method name is camelCase and properly capitalized.

6. Class variables Ok. Every class variable is in mixed case and properly capitalized.

7. Constants The field

```
1  private static final module
```

and

```
1  private static final instanceId
```

are immutable, so they can be considered as constant; they should be capitalized. Instead

```
1  private static final registeredManagers
```

is fine because it's mutable

3.2 Indentation

8. Number of spaces Ok, the code is consistently indented with 4 spaces.

9. No tabs for indentation Ok. No tabs are used to indent the code.

3.3 Braces

10. Consistent bracing style Ok. The code is consistently braced following the *"Kernighan and Ritchie"* style.

11. One-line statements bracing No. In the code we can find the following statements:

```
1•      if (Debug.infoOn()) Debug.logInfo("Scheduling Job : " + job,
           module);
```

on line 326;

```
1•      if (Debug.infoOn())
2       Debug.logInfo("-- " + rescheduled + " jobs re-scheduled",
           module);
```

on line 351-352;

```
1•      if (Debug.infoOn())
2       Debug.logInfo("No crashed jobs to re-schedule", module);
```

on line 354-355;

3.4 File Organization

12. Blank lines as separation   Ok. Blank lines are present between each method, around imports and variable declarations. Most of the methods also begin with a Javadoc.

13. Where practical, line length under 80 characters   NOPE righe 73, 74, 89, 126, 147, 150, 154, 156, 161, 182, 186-190, 195, 198, 201, A great number of lines exceed 80 characters

14. Line length always under 120 characters   NEPPURE righe 74, 186, 198, 217, 221, 222, 261-264, 273, 311, 315, 317, 387, 409, 429, 453, 498, 543, 560, 561 le dichiarazioni dei metodi sono lunghissime e wrappate poco

3.5 Wrapping Lines

15. Line breaks after commas and operators   No. riga 152, la virgola dovrebbe stare sopra

16. Higher-level breaks are used   Ok. There aren't any line break with operators

17. Statements are aligned to previous ones   Ok. All statements are aligned to previous ones.

3.6 Comments

18. Comments use   The method

```
1      public synchronized void reloadCrashedJobs()
```

on line 305 is not commented and so it isn't easy to understand.
The rest of the code is well commented.

19. Commented out code   There aren't lines of code hidden in block of comments in the source code.

3.7 Java Source Files

20. Single public class or interface   Ok. Job manager is the only public class declared in the file. There are no other classes.

21. The public class is the first class in the file   Ok. Job manager is the only public class declared in the file. There are no other classes.

22. External program interfaces are consistent with the Javadoc Ok
Quello che c'è scritto nella prima javadoc viene rispettato. la Javadoc parla di
runJob e schedule

abbiamo anche altri metodi pubblici: getter: - getDelegator - getDispatcher
- getInstance - getPoolState poi altre robe - isAvailable - reloadCrashedJobs -
poll

forse sarebbe una buona idea aggiungere anche reloadCrashedJobs e poll alla
javadoc

23. The Javadoc is complete NO.

- No javadoc for 'module'! line 71

- No javadoc for 'instanceId'! line 71

- No javadoc for reloadCrashedJob!! line 304

- Missing @return tag on getInstance, line 88

- Missing @return tag on getDelegator, line 119

- Missing @return tag on getDispatcher, line 124

- Missing @param tag for 'limit' on poll, line 174

- Missing @return tag on poll, line 174

- Missing @param tag for 'job' on runJob, line 363

- Missing @throws tag for 'JobManagerException' on runJob, line 363

- Missing @throws tag for 'JobManagerException' on schedule, line 386,
  408, 428, 453, 469, 498, 543

assertIsRunning, getRunPools sono private quindi non hanno necessaria-
mente bisogno di javadoc

3.8 Package and Import Statements

24. Package statements are first, import statements second Ok. One
package statements. All import statements immediately follow.

3.9 Class and Interface Declarations

25. The class declarations should follow a specific order - javadoc ok -
class declaration ok - altri commenti / - static variables ok - public ok - private
ok - normal variables - constructors - methods

No, abbiamo variabili statiche, poi un po' di metodi statici, poi variabili
normali, poi costruttori (getInstance è un costruttore), setter e getter poi un
metodo statico

-> i metodi statici iniziali andrebbero spostati dopo i costruttori

26. Methods are grouped by functionality Ok. i gruppi di funzionalita'
sono:

check status e costruttori: assertIsRunning getInstance shutDown

getter vari: getDelegator getDispatcher getPoolState

funzionamento dei jobs: isAvailable getRunPools pool reloadCrashedJobs
runJob schedule

**27. The code is free of duplicates, long methods, big classes, breaking
encapsulation, and coupling and cohesion are adequate** small class
duplicates? no short methods no breaking encapsulation

low/loose coupling -> ci sono un sacco di delegator e dispatcher high cohesion
-> tutti i metodi servono a runnare/queuare jobs

3.10   Initialization and Declarations

28. Visibility All variables and class members are of the correct type and
have the proper visibility. In line 305 the method

```
1  public synchronized void reloadCrashedJobs()
```

could be stated as protected.

29. Proper scope. OK. All variables are declared in the proper scope

30. New objects. OK. Each time a new object is desired the proper con-
structor is called

31. All object references are initialized before use. OK. All reference
are initialized before any object uses it.

32. Variables initialization. OK. All variables are initialized where they
are declared, unless dependent upon a computation.

33. Declarations. OK. Each declaration appear at the beginning of blocks.

3.11   Method Calls

34. Correct orders parameters Sembra tutto bene

35. The called method is the right method Sembra di si

36. The returned value from the method is used properly Me pare de
si

3.12 Arrays

37. No off-by-one errors in array indexing Ok. The only indexing is with foreach, so there can't be off-by-one errors.

38. No out-of-bounds indexes Ok. There is no number indexing.

39. Constructors are called when a new array item is desired Ok. There are no arrays, only collections. Every collection is created with the appropriate constructor.

3.13 Object Comparison

40. Objects are compared with equals Ok. There are no object comparisons.

3.14 Output Format

41. Displayed output is free of spelling and grammatical errors 2 minor mistakes:
    riga 156: Debug.logWarning(e, "Exception thrown while check lock on JobManager : " + instanceId, module); dovrebbe essere "while checking"
    riga 182: Debug.logWarning("Unable to locate DispatchContext object; not running job!", module); dovrebbe essere "job:", come negli altri log di debug

42. Error messages are comprehensive and useful ok, all error messages clearly explain what type of problem has occurred

43. Output is formatted correctly in terms of line breaks and spacing Quasi ok, there are no outputs that need line breaks BUT Some debug outputs don't have a trailing space (spazio alla fine)

3.15 Computation, Comparisons and Assignments

44. "Brutish programming". the avoids OK. The implementation avoids brute force solutions; the code is simple and concise.

45. Operator precedence and parenthesizing. OK. Computation/evaluation of operator precedence and parentheses is in the proper order.

46. The liberal use of parenthesis is used to avoid operator precedence problems. OK. There aren't any parenthesis used in an appropriate way.

47. All denominators of a division are prevented from being zero. OK. There are no division.

48. Integer arithmetic, especially division, are used appropriately to avoid causing unexpected truncation/rounding.   OK. Integer arithmetic is used only to increment variable.

49. Comparison and Boolean operators are correct.   OK.

50. Throw-catch expressions.   OK. The error condition is always legitimate

51. The code is free of any implicit type conversions.   OK. The code is free

3.16   Exceptions

52. Relevant exceptions are caught.   OK.

53. The appropriate action is taken for each catch block.   OK. There are two general

```
1  catch (Throwable t)
```

in order to guarantee a working jobPoller even when a database connection is not available.

3.17   Flow of Control

54. All switch cases are addressed with a break   Ok, no switch statements.

55. All switch statements have a default branch   Ok, no switch statements.

56. All loops are correctly formed, with appropriate initialization, increments and termination expressions   Ok. All for loops are foreach, so no checks on bounds are needed

Ok anche per il while loop at line 219: GenericValue jobValue = jobsIterator.next(); while (jobValue != null) jobValue = jobIterator.next(); tutto ok, l'iteratore va avanti finche' non finiscono i valori, poi esce dal while

while a riga 275 the same

3.18   Files

The JobManager class does not have to handle files.

4   Other Problems

The assigned class does not contain any other relevant problem.
For the sake of accuracy, we suggest to substitute in line 576

```
1  jFields.put("currentRetryCount", new Long(0));
```

with the more efficient

```
1   jFields.put("currentRetryCount", valueOf(0));
```

Using new Long() is guaranteed to always result in a new object whereas Integer.valueOf() allows caching of values to be done by the compiler, class library, or JVM. Using of cached values avoids object allocation and the code will be faster.
This might have to be considered in case of multiple new Long() calls.

5   Effort Spent

- Pietro Ferretti: hours of work

- Nicole Gervasoni: hours of work

- Danilo Labanca: hours of work

6   Revisions

6.1   Changelog

- CID v1.0, published on February 5, 2017