Politecnico di Milano, A.Y. 2016/2017
M.Sc. Degree Programme in Computer Science and Engineering
Software Engineering 2 Project

bCode Insepection Document - Apache OFbiz

Philippe Scorsolini,
Lorenzo Semeria,
Gabriele Vanoni

4th February 2017

Contents

1 Class assigned

In this document we'll inspect the code of the class "XmlSerializer" of the project "Apache OFBiz®". OFBiz is an Enterprise Resource Planning (ERP) System written in Java. Our class is part of the `entity.serialize` package, a very small package that only contains our class, the interface `XmlSerializable` and the definition of an `Exception`, `SerializeException` which extends an OFBiz Exception.
OFBiz uses Gradle as Build Tool and therefore its convention. The class we will analyze is located at this address:
`../apache-ofbiz-16.11.01/framework/entity/src/main/java/org/apache/ofbiz/entity/serialize/XmlSerializer.java`.

2 Functional role of assigned class

The assigned class is deprecated, as clearly stated in the starting `javadoc` comment. Its purpose is to handle the serialization of objects into XML and vice versa.
The method `serialize` takes an `Object` and returns a `String`, which is the resulting XML for the provided `Object`. The other methods used for serialization, namely `serializeSingle` and `serializeCustom` return an `Element` which is an instance of `Node` from package `org.w3c.dom`, part of the standard Java library. They both take as input an `Object` to serialize and a `Document` object.
All deserialization methods, as expected, return an `Object` and take as input either an XML `Element` or a whole `Document`, whose definitions are found in `org.w3c.dom`.

3 List of issues found by applying the checklist

3.1 Notation Used

We will adopt the following notation to simplify reading the document:

- **L. 12** to indicate a single line (line 12 in this case).
- **L. 12, 15** to indicate a list of non consevutive lines (lines 12 and 15 in this case).
- **L. 12-34** to indicate an interval of lines (line 12 through 34 in this case).
- ***C12*** to indicate the 12-th element of the provided checklist.

3.2 Issues

- ***C11***. "All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces." **L. 295, 433, 446**
- ***C14***. "When line length must exceed 80 characters, it does NOT exceed 120 characters." **L. 176, 217, 350, 446, 448**
- ***C18***. "Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing." not all methods are commented appropriately, there is an overall good usage of comments, but some methods could have been more clearly explained.
- ***C19***. "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." **L. 171, 194**
- ***C23***. "Check that the javadoc is complete". **L. 131, 259, 276, 292, 464** contain `public` methods that do not have any javadoc. The other methods have some javadoc but no straightforward or detailed explanation is given.
- ***C25 (c).*** "class/interface implementation comment, if necessary;" No such comment is provided, while it would have been useful to have a general idea of what the class is for. This may or may not be due to the fact that the class-level javadoc has been replaced by a deprecated notice.
- ***C27.*** "Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate". In **L. 73** the throws are redundant: `FileNotFoundException extends IOException` but both are present in the `throws` statement. The method at **L. 292** is close to 200 lines long, which make it hard to keep track of the code. The method is mainly composed of `if`s and `else if`s so knowing which branch a part of code is in is very hard since there are so many cases.
- ***C28***. "[...] Check that they have the right visibility (public/private/protected).". While the methods at **L. 93, 118, 131** are used throughout the package, methods at **L. 259, 276, 292, 464** are never used except internally in this class. They are likely helper methods for this class's functionality but they do not seem to be useful outside of this, therefore they should be `private`.

- ***C29***. "Check that variables are declared in the proper scope.". Variable at **L. 69** is `public` but is not accessed in the whole project. Its purpose is not clear but should probably be set to `private`.
- ***C30***. "Check that constructors are called when a new object is desired.". Throughout the code it almost always happens that objects are created using external methods. While this clashes with the given entry in the checklist, it seems a correct choice to call external methods to partially process data and create the proper object.
- ***C31***. "Check that all object references are initialized before use.". At **L. 492** the variable `formatter` is initialized to `null`, which is the default value for every `Object`. However, it is impossible for this variable to be `null` at **L. 501** (`return` statement) since it is either initialized at **L. 495** or at **L. 498**.
- ***C33***. "Declarations appear at the beginning of blocks". At **L. 196-198** variables are declared in the middle of the `else if` block opened at **L. 172**. Same happens at **L. 233-235**, variables are declared in the `else if` block opened at **L. 213**. In `makeElement` at **L. 276-290**, variable element is created twice with different visibility. At **L. 286** it is created but not in the beginning of the containing block.
- ***C36***. "Check that method returned values are used properly.". In many occasions, for instance when calling `appendChild` from `org.w3c.dom.Node.java` in **L. 201** (and many others) the return value is not used. However, since the children is added to the `Node` object regardless of the returned value, it may not be necessary.
- ***C60***. "Check that all file exceptions are caught and dealt with accordingly". **L73** FileNotFoundException is thrown instead of caught.

4 Other issues

Throughout the code and in particular in blocks at **L. 131-257** and **L.292-462** there is a series of `if...else if...else if...` statements. This way of programming is not compliant with the "open/closed principle", one of the master ideas of object oriented programming, that states "software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification". In fact this class is not "open" for extension. Even worse in the first block mentioned the condition inside `if` statements is an `instance of` call, a witness of bad usage of overriding.

5 Effort spent

Component	Time spent (in hour)
Philippe Scorsolini	10
Lorenzo Semeria	12
Gabriele Vanoni	15