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

Code Insepection Document - Apache OFbiz

Philippe Scorsolini,
Lorenzo Semeria,
Gabriele Vanoni

5th 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	6
Lorenzo Semeria	8
Gabriele Vanoni	6

A. Source Code

```
1  /*******************************************************************************
2   * Licensed to the Apache Software Foundation (ASF) under one
3   * or more contributor license agreements. See the NOTICE file
4   * distributed with this work for additional information
5   * regarding copyright ownership. The ASF licenses this file
6   * to you under the Apache License, Version 2.0 (the
7   * "License"); you may not use this file except in compliance
8   * with the License. You may obtain a copy of the License at
9   *
10  * http://www.apache.org/licenses/LICENSE-2.0
11  *
12  * Unless required by applicable law or agreed to in writing,
13  * software distributed under the License is distributed on an
14  * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
15  * KIND, either express or implied. See the License for the
16  * specific language governing permissions and limitations
17  * under the License.
18  *******************************************************************************/

19 package org.apache.ofbiz.entity.serialize;
20
21 import java.io.FileNotFoundException;
22 import java.io.IOException;
23 import java.io.Serializable;
24 import java.lang.ref.WeakReference;
25 import java.math.BigDecimal;
26 import java.text.DateFormat;
27 import java.text.ParseException;
28 import java.text.SimpleDateFormat;
29 import java.util.ArrayList;
30 import java.util.Calendar;
31 import java.util.Collection;
32 import java.util.HashMap;
33 import java.util.HashSet;
34 import java.util.Hashtable;
35 import java.util.Iterator;
36 import java.util.LinkedList;
37 import java.util.Locale;
38 import java.util.Map;
39 import java.util.Properties;
40 import java.util.Stack;
41 import java.util.TreeMap;
42 import java.util.TreeSet;
43 import java.util.Vector;
44 import java.util.WeakHashMap;
45
```

```
46 import javax.xml.bind.DatatypeConverter;
47 import javax.xml.parsers.ParserConfigurationException;
48
49 import org.apache.ofbiz.base.util.Debug;
50 import org.apache.ofbiz.base.util.StringUtil;
51 import org.apache.ofbiz.base.util.UtilGenerics;
52 import org.apache.ofbiz.base.util.UtilMisc;
53 import org.apache.ofbiz.base.util.UtilObject;
54 import org.apache.ofbiz.base.util.UtilXml;
55 import org.apache.ofbiz.entity.Delegator;
56 import org.apache.ofbiz.entity.GenericPK;
57 import org.apache.ofbiz.entity.GenericValue;
58 import org.w3c.dom.Document;
59 import org.w3c.dom.Element;
60 import org.w3c.dom.Node;
61 import org.xml.sax.SAXException;
62
63 /**
64  * XmlSerializer class. This class is deprecated - new code should use
       the
65  * Java object marshalling/unmarshalling methods in <code>UtilXml.java</
       code>.
66  *
67  */
68 public class XmlSerializer {
69     public static final String module = XmlSerializer.class.getName();
70
71     private static WeakReference<DateFormat> simpleDateFormatter;
72
73     public static String serialize(Object object) throws
           SerializeException, FileNotFoundException, IOException {
74         Document document = UtilXml.makeEmptyXmlDocument("ofbiz-ser");
75         Element rootElement = document.getDocumentElement();
76
77         rootElement.appendChild(serializeSingle(object, document));
78         return UtilXml.writeXmlDocument(document);
79     }
80
81     /** Deserialize a Java object from an XML string. <p>This method
           should be used with caution.
82      * If the XML string contains a serialized <code>GenericValue</code>
            or <code>GenericPK</code>
83      * then it is possible to unintentionally corrupt the database.</p>
84      *
85      * @param content the content
86      * @param delegator the delegator
87      * @return return a deserialized object from XML string
88      * @throws SerializeException
89      * @throws SAXException
```

```
90       * @throws ParserConfigurationException
91       * @throws IOException
92       */
93      public static Object deserialize(String content, Delegator delegator)
94          throws SerializeException, SAXException,
                ParserConfigurationException, IOException {
95          // readXmlDocument with false second parameter to disable
                validation
96          Document document = UtilXml.readXmlDocument(content, false);
97          if (document != null) {
98              if (!"ofbiz-ser".equals(document.getDocumentElement().
                    getTagName())) {
99                  return UtilXml.fromXml(content);
100             }
101             return deserialize(document, delegator);
102         } else {
103             Debug.logWarning("Serialized document came back null", module)
                    ;
104             return null;
105         }
106     }
107
108     /** Deserialize a Java object from a DOM <code>Document</code>.
109      * <p>This method should be used with caution. If the DOM <code>
             Document</code>
110      * contains a serialized <code>GenericValue</code> or <code>GenericPK
             </code>
111      * then it is possible to unintentionally corrupt the database.</p>
112      *
113      * @param document the document
114      * @param delegator the delegator
115      * @return returns a deserialized object from a DOM document
116      * @throws SerializeException
117      */
118     public static Object deserialize(Document document, Delegator
            delegator) throws SerializeException {
119         Element rootElement = document.getDocumentElement();
120         // find the first element below the root element, that should be
                the object
121         Node curChild = rootElement.getFirstChild();
122         while (curChild != null && curChild.getNodeType() != Node.
                ELEMENT_NODE) {
123             curChild = curChild.getNextSibling();
124         }
125         if (curChild == null) {
126             return null;
127         }
128         return deserializeSingle((Element) curChild, delegator);
129     }
```

```
130
131     public static Element serializeSingle(Object object, Document
            document) throws SerializeException {
132         if (document == null) return null;
133
134         if (object == null) return makeElement("null", object, document);
135
136         // - Standard Objects -
137         if (object instanceof String) {
138             return makeElement("std-String", object, document);
139         } else if (object instanceof Integer) {
140             return makeElement("std-Integer", object, document);
141         } else if (object instanceof Long) {
142             return makeElement("std-Long", object, document);
143         } else if (object instanceof Float) {
144             return makeElement("std-Float", object, document);
145         } else if (object instanceof Double) {
146             return makeElement("std-Double", object, document);
147         } else if (object instanceof Boolean) {
148             return makeElement("std-Boolean", object, document);
149         } else if (object instanceof Locale) {
150             return makeElement("std-Locale", object, document);
151         } else if (object instanceof BigDecimal) {
152             String stringValue = ((BigDecimal) object).setScale(10,
                    BigDecimal.ROUND_HALF_UP).toString();
153             return makeElement("std-BigDecimal", stringValue, document);
154             // - SQL Objects -
155         } else if (object instanceof java.sql.Timestamp) {
156             String stringValue = object.toString().replace(' ', 'T');
157             return makeElement("sql-Timestamp", stringValue, document);
158         } else if (object instanceof java.sql.Date) {
159             return makeElement("sql-Date", object, document);
160         } else if (object instanceof java.sql.Time) {
161             return makeElement("sql-Time", object, document);
162         } else if (object instanceof java.util.Date) {
163             // NOTE: make sure this is AFTER the java.sql date/time
                    objects since they inherit from java.util.Date
164             DateFormat formatter = getDateFormat();
165             String stringValue = null;
166
167             synchronized (formatter) {
168                 stringValue = formatter.format((java.util.Date) object);
169             }
170             return makeElement("std-Date", stringValue, document);
171             // return makeElement("std-Date", object, document);
172         } else if (object instanceof Collection<?>) {
173             // - Collections -
174             String elementName = null;
175
```

```
176             // these ARE order sensitive; for instance Stack extends
                    Vector, so if Vector were first we would lose the stack
                    part
177             if (object instanceof ArrayList<?>) {
178                 elementName = "col-ArrayList";
179             } else if (object instanceof LinkedList<?>) {
180                 elementName = "col-LinkedList";
181             } else if (object instanceof Stack<?>) {
182                 elementName = "col-Stack";
183             } else if (object instanceof Vector<?>) {
184                 elementName = "col-Vector";
185             } else if (object instanceof TreeSet<?>) {
186                 elementName = "col-TreeSet";
187             } else if (object instanceof HashSet<?>) {
188                 elementName = "col-HashSet";
189             } else {
190                 // no specific type found, do general Collection, will
                        deserialize as LinkedList
191                 elementName = "col-Collection";
192             }
193
194             // if (elementName == null) return serializeCustom(object,
                    document);
195
196             Collection<?> value = UtilGenerics.cast(object);
197             Element element = document.createElement(elementName);
198             Iterator<?> iter = value.iterator();
199
200             while (iter.hasNext()) {
201                 element.appendChild(serializeSingle(iter.next(), document)
                        );
202             }
203             return element;
204         } else if (object instanceof GenericPK) {
205             // Do GenericEntity objects as a special case, use std XML
                    import/export routines
206             GenericPK value = (GenericPK) object;
207
208             return value.makeXmlElement(document, "eepk-");
209         } else if (object instanceof GenericValue) {
210             GenericValue value = (GenericValue) object;
211
212             return value.makeXmlElement(document, "eeval-");
213         } else if (object instanceof Map<?, ?>) {
214             // - Maps -
215             String elementName = null;
216
217             // these ARE order sensitive; for instance Properties extends
                     Hashtable, so if Hashtable were first we would lose the
```

```
                Properties part
218             if (object instanceof HashMap<?, ?>) {
219                 elementName = "map-HashMap";
220             } else if (object instanceof Properties) {
221                 elementName = "map-Properties";
222             } else if (object instanceof Hashtable<?, ?>) {
223                 elementName = "map-Hashtable";
224             } else if (object instanceof WeakHashMap<?, ?>) {
225                 elementName = "map-WeakHashMap";
226             } else if (object instanceof TreeMap<?, ?>) {
227                 elementName = "map-TreeMap";
228             } else {
229                 // serialize as a simple Map implementation if nothing
                        else applies, these will deserialize as a HashMap
230                 elementName = "map-Map";
231             }
232
233             Element element = document.createElement(elementName);
234             Map<?,?> value = UtilGenerics.cast(object);
235             Iterator<Map.Entry<?, ?>> iter = UtilGenerics.cast(value.
                    entrySet().iterator());
236
237             while (iter.hasNext()) {
238                 Map.Entry<?,?> entry = iter.next();
239
240                 Element entryElement = document.createElement("map-Entry")
                        ;
241
242                 element.appendChild(entryElement);
243
244                 Element key = document.createElement("map-Key");
245
246                 entryElement.appendChild(key);
247                 key.appendChild(serializeSingle(entry.getKey(), document))
                        ;
248                 Element mapValue = document.createElement("map-Value");
249
250                 entryElement.appendChild(mapValue);
251                 mapValue.appendChild(serializeSingle(entry.getValue(),
                        document));
252             }
253             return element;
254         }
255
256         return serializeCustom(object, document);
257     }
258
259     public static Element serializeCustom(Object object, Document
            document) throws SerializeException {
```

```
260         if (object instanceof Serializable) {
261             byte[] objBytes = UtilObject.getBytes(object);
262             if (objBytes == null) {
263                 throw new SerializeException("Unable to serialize object;
                        null byte array returned");
264             } else {
265                 String byteHex = StringUtil.toHexString(objBytes);
266                 Element element = document.createElement("cus-obj");
267                 // this is hex encoded so does not need to be in a CDATA
                        block
268                 element.appendChild(document.createTextNode(byteHex));
269                 return element;
270             }
271         } else {
272             throw new SerializeException("Cannot serialize object of
                    class " + object.getClass().getName());
273         }
274     }
275
276     public static Element makeElement(String elementName, Object value,
            Document document) {
277         if (value == null) {
278             Element element = document.createElement("null");
279             element.setAttribute("xsi:nil", "true");
280             // I tried to put the schema in the envelope header (in
                    createAndSendSOAPResponse)
281             // resEnv.declareNamespace("http://www.w3.org/2001/XMLSchema-
                    instance", null);
282             // But it gets prefixed and that does not work. So adding in
                    each instance
283             element.setAttribute("xmlns:xsi", "http://www.w3.org/2001/
                    XMLSchema-instance");
284             return element;
285         }
286         Element element = document.createElement(elementName);
287
288         element.setAttribute("value", value.toString());
289         return element;
290     }
291
292     public static Object deserializeSingle(Element element, Delegator
            delegator) throws SerializeException {
293         String tagName = element.getLocalName();
294
295         if (tagName.equals("null")) return null;
296
297         if (tagName.startsWith("std-")) {
298             // - Standard Objects -
299             if ("std-String".equals(tagName)) {
```

```
300             return element.getAttribute("value");
301         } else if ("std-Integer".equals(tagName)) {
302             String valStr = element.getAttribute("value");
303             return Integer.valueOf(valStr);
304         } else if ("std-Long".equals(tagName)) {
305             String valStr = element.getAttribute("value");
306             return Long.valueOf(valStr);
307         } else if ("std-Float".equals(tagName)) {
308             String valStr = element.getAttribute("value");
309             return Float.valueOf(valStr);
310         } else if ("std-Double".equals(tagName)) {
311             String valStr = element.getAttribute("value");
312             return Double.valueOf(valStr);
313         } else if ("std-BigDecimal".equals(tagName)) {
314             String valStr = element.getAttribute("value");
315             return new BigDecimal(valStr);
316         } else if ("std-Boolean".equals(tagName)) {
317             String valStr = element.getAttribute("value");
318             return Boolean.valueOf(valStr);
319         } else if ("std-Locale".equals(tagName)) {
320             String valStr = element.getAttribute("value");
321             return UtilMisc.parseLocale(valStr);
322         } else if ("std-Date".equals(tagName)) {
323             String valStr = element.getAttribute("value");
324             DateFormat formatter = getDateFormat();
325             java.util.Date value = null;
326 
327             try {
328                 synchronized (formatter) {
329                     value = formatter.parse(valStr);
330                 }
331             } catch (ParseException e) {
332                 throw new SerializeException("Could not parse date
                        String: " + valStr, e);
333             }
334             return value;
335         }
336     } else if (tagName.startsWith("sql-")) {
337         // - SQL Objects -
338         if ("sql-Timestamp".equals(tagName)) {
339             String valStr = element.getAttribute("value");
340             /*
341              * sql-Timestamp is defined as xsd:dateTime in
                     ModelService.getTypes(),
342              * so try to parse the value as xsd:dateTime first.
343              * Fallback is java.sql.Timestamp because it has been this
                    way all the time.
344              */
345             try {
```

```
346                 Calendar cal = DatatypeConverter.parseDate(valStr);
347                 return new java.sql.Timestamp(cal.getTimeInMillis());
348             }
349             catch (Exception e) {
350                 Debug.logWarning("sql-Timestamp does not conform to
                        XML Schema definition, try java.sql.Timestamp
                        format", module);
351                 return java.sql.Timestamp.valueOf(valStr);
352             }
353         } else if ("sql-Date".equals(tagName)) {
354             String valStr = element.getAttribute("value");
355             return java.sql.Date.valueOf(valStr);
356         } else if ("sql-Time".equals(tagName)) {
357             String valStr = element.getAttribute("value");
358             return java.sql.Time.valueOf(valStr);
359         }
360     } else if (tagName.startsWith("col-")) {
361         // - Collections -
362         Collection<Object> value = null;
363 
364         if ("col-ArrayList".equals(tagName)) {
365             value = new ArrayList<Object>();
366         } else if ("col-LinkedList".equals(tagName)) {
367             value = new LinkedList<Object>();
368         } else if ("col-Stack".equals(tagName)) {
369             value = new Stack<Object>();
370         } else if ("col-Vector".equals(tagName)) {
371             value = new Vector<Object>();
372         } else if ("col-TreeSet".equals(tagName)) {
373             value = new TreeSet<Object>();
374         } else if ("col-HashSet".equals(tagName)) {
375             value = new HashSet<Object>();
376         } else if ("col-Collection".equals(tagName)) {
377             value = new LinkedList<Object>();
378         }
379 
380         if (value == null) {
381             return deserializeCustom(element);
382         } else {
383             Node curChild = element.getFirstChild();
384 
385             while (curChild != null) {
386                 if (curChild.getNodeType() == Node.ELEMENT_NODE) {
387                     value.add(deserializeSingle((Element) curChild,
                            delegator));
388                 }
389                 curChild = curChild.getNextSibling();
390             }
391             return value;
```

```
392             }
393         } else if (tagName.startsWith("map-")) {
394             // - Maps -
395             Map<Object, Object> value = null;
396 
397             if ("map-HashMap".equals(tagName)) {
398                 value = new HashMap<Object, Object>();
399             } else if ("map-Properties".equals(tagName)) {
400                 value = new Properties();
401             } else if ("map-Hashtable".equals(tagName)) {
402                 value = new Hashtable<Object, Object>();
403             } else if ("map-WeakHashMap".equals(tagName)) {
404                 value = new WeakHashMap<Object, Object>();
405             } else if ("map-TreeMap".equals(tagName)) {
406                 value = new TreeMap<Object, Object>();
407             } else if ("map-Map".equals(tagName)) {
408                 value = new HashMap<Object, Object>();
409             }
410 
411             if (value == null) {
412                 return deserializeCustom(element);
413             } else {
414                 Node curChild = element.getFirstChild();
415 
416                 while (curChild != null) {
417                     if (curChild.getNodeType() == Node.ELEMENT_NODE) {
418                         Element curElement = (Element) curChild;
419 
420                         if ("map-Entry".equals(curElement.getLocalName()))
                             {
421 
422                             Element mapKeyElement = UtilXml.
                                 firstChildElement(curElement, "map-Key");
423                             Element keyElement = null;
424                             Node tempNode = mapKeyElement.getFirstChild();
425 
426                             while (tempNode != null) {
427                                 if (tempNode.getNodeType() == Node.
                                     ELEMENT_NODE) {
428                                     keyElement = (Element) tempNode;
429                                     break;
430                                 }
431                                 tempNode = tempNode.getNextSibling();
432                             }
433                             if (keyElement == null) throw new
                                 SerializeException("Could not find an
                                 element under the map-Key");
434 
```

```
435                         Element mapValueElement = UtilXml.
                                firstChildElement(curElement, "map-Value");
436                         Element valueElement = null;
437
438                         tempNode = mapValueElement.getFirstChild();
439                         while (tempNode != null) {
440                             if (tempNode.getNodeType() == Node.
                                    ELEMENT_NODE) {
441                                 valueElement = (Element) tempNode;
442                                 break;
443                             }
444                             tempNode = tempNode.getNextSibling();
445                         }
446                         if (valueElement == null) throw new
                                SerializeException("Could not find an
                                element under the map-Value");
447
448                         value.put(deserializeSingle(keyElement,
                                delegator), deserializeSingle(valueElement,
                                delegator));
449                     }
450                 }
451                 curChild = curChild.getNextSibling();
452             }
453             return value;
454         }
455     } else if (tagName.startsWith("eepk-")) {
456         return delegator.makePK(element);
457     } else if (tagName.startsWith("eeval-")) {
458         return delegator.makeValue(element);
459     }
460
461     return deserializeCustom(element);
462 }
463
464 public static Object deserializeCustom(Element element) throws
        SerializeException {
465     String tagName = element.getLocalName();
466     if ("cus-obj".equals(tagName)) {
467         String value = UtilXml.elementValue(element);
468         if (value != null) {
469             byte[] valueBytes = StringUtil.fromHexString(value);
470             if (valueBytes != null) {
471                 Object obj = UtilObject.getObject(valueBytes);
472                 if (obj != null) {
473                     return obj;
474                 }
475             }
476         }
```

```
477             throw new SerializeException("Problem deserializing object
                    from byte array + " + element.getLocalName());
478         } else {
479             throw new SerializeException("Cannot deserialize element
                    named " + element.getLocalName());
480         }
481     }
482
483     /**
484      * Returns the DateFormat used to serialize and deserialize <code>
             java.util.Date</code> objects.
485      * This format is NOT used to format any of the java.sql subtypes of
             java.util.Date.
486      * A <code>WeakReference</code> is used to maintain a reference to
             the DateFormat object
487      * so that it can be created and garbage collected as needed.
488      *
489      * @return the DateFormat used to serialize and deserialize <code>
             java.util.Date</code> objects.
490      */
491     private static DateFormat getDateFormat() {
492         DateFormat formatter = null;
493
494         if (simpleDateFormatter != null) {
495             formatter = simpleDateFormatter.get();
496         }
497         if (formatter == null) {
498             formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.S");
499             simpleDateFormatter = new WeakReference<DateFormat>(formatter)
                    ;
500         }
501         return formatter;
502     }
503 }
```