PowerEnJoy
Code Inspection Document
Software Engineering 2 project

POLITECNICO
MILANO 1863

Authors:
Arcari Leonardo
Bertoglio Riccardo
Galimberti Andrea

February 5, 2017

Document version: 1.0

Contents

1 Code Description

1.1 Assigned Class

Our group was assigned the task of analyzing the *XmlWidgetVisitor* class of the *Apache OFBiz* project; the assigned class is located in the *org.apache.ofbiz.widget.model* package.

The code inspection of the class has been performed with regards to the checklist provided in the *Code Inspection Assignment Task Description* document; also, other issues not in the checklist were additionally looked for.

1.2 Functional Role of the Assigned Class

The functional role of *XmlWidgetVisitor* is to generate an XML representation of a *ModelWidget* class. A *ModelWidget* is an abstract base class for derived graphical widgets. The role of *XmlWidgetVisitor*, as an implementation of a *ModelWidgetVisitor* is to parse a widget and its subwidgets and serialize them into a structured, textual representation according to the XML standard.

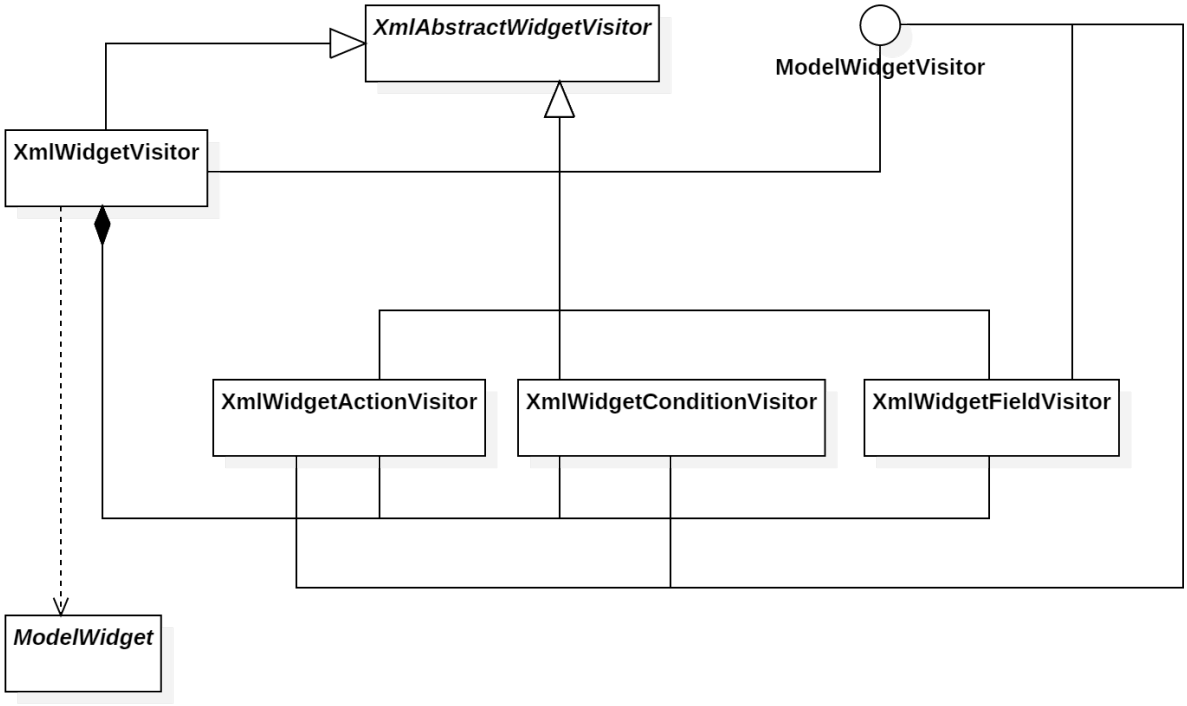
XmlWidgetVisitor writes the XML code onto an output stream passed as a parameter to the class constructor.

Each method **implements** the corresponding one exposed by the *ModelWidgetVisitor* interface, which are **overloads** of the same method with different type of input parameter.

Such information could be gathered by reading the class header *javadoc*:

> *An object that generates an XML representation from widget models.*
> *The generated XML is unformatted - if you want to "pretty print" the XML, then use a transformer*

Also, to achieve a better knowledge of the classes *XmlWidgetVisitor* has to deal with, it was required to navigate through the code of their implementation in order to discover the classes hierarchy and their functional role as well. Following a class diagram to graphically show the claims made above:


XmlAbstractWidgetVisitor
ModelWidgetVisitor
XmlWidgetVisitor
XmlWidgetActionVisitor
XmlWidgetConditionVisitor
XmlWidgetFieldVisitor
ModelWidget

2 Code Inspection

For each rule from the checklist, the result of the code inspection is provided and, in case any issue was found, the concerned line of code is possibly specified.

In the following sections, single lines of code are denoted as *L. 111,* multiple lines of code are denoted as *L. 111, 222* and intervals of lines of code are denoted as *L. 111-222.*

2.1 Naming Conventions

2.1.1 Rule 1

All class names, interface names, method names, class variables, method variables, and constants have sufficiently meaningful names and do what the name suggests, hence the rule is respected.

2.1.2 Rule 2

One-character variables are never used, not even for temporary "throwaway" variables, so the rule is correctly followed.

2.1.3 Rule 3

All the used class names are nouns, in mixed case, with the first letter of each word capitalized, hence the rule is respected.

2.1.4 Rule 4

The interface implemented by the class has a name capitalized like other classes, and no other interface is defined, so the rule is respected.

2.1.5 Rule 5

Method names are verbs, with the first letter of each additional word capitalized, therefore the rule is abided by.

2.1.6 Rule 6

All the class variables start with a lowercase first letter, with the first letter of each additional word capitalized, therefore the rule is abided by.

2.1.7 Rule 7

No constant is declared, so the rule is correctly followed.

2.2 Indention

2.2.1 Rule 8

Four spaces are always used for indentation, hence the rule is respected.

2.2.2 Rule 9

No tabs are ever used to indent, hence the rule is respected.

2.3 Braces

2.3.1 Rule 10

Kernighan and Ritchie style is always used, hence the rule is abided by.

2.3.2 Rule 11

The rule is respected, there isn't any if, while, do-while, try-catch, or for statement with only one statement to execute that isn't surrounded by curly braces.

2.4 File Organization

2.4.1 Rule 12

- L. 19 should be blank to separate beginning comment section from package statement and should be moved to the next line.

2.4.2 Rule 13

- L. 56 exceeds 80 characters while implementation statement could be moved to the following line. In fact interfaces implementation are important to understand the class behavior and should be readable without scrolling the class text.
- L. 456 exceeds 80 characters due to a *for-each* statement iterating over an *Iterable* returned by a method invocation chaining. Method invocation chaining could be split on two or three lines.

2.4.3 Rule 14

No line exceeds 120 characters.

2.5 Wrapping Lines

2.5.1 Rule 15

No line breaks.

2.5.2 Rule 16

No line breaks.

2.5.3 Rule 17

A new statement is aligned with the beginning of the expression at the same level as the previous line.

2.6 Comments

2.6.1 Rule 18

Only the purpose of the class is briefly commented, there isn't any other comment for any block of code or method inside it. There should be many more comments, the rule is definitely not abided by.

2.6.2 Rule 19

No line of code is commented out, so the rule is respected.

2.7 Java Source Files

2.7.1 Rule 20

The source file contains only one public class, hence the rule is respected.

2.7.2 Rule 21

The public class is the first one (also, actually there aren't any other classes contained in the source file), hence the rule is respected.

2.7.3 Rule 22

External program interfaces can actually be said to be implemented consistently with what is described in the OFBiz Javadoc, as the *ModelWidgetVisitor* interface, which is the only one implemented by the *XmlWidgetVisitor*, is seriously lacking and has no explanation at all of its purpose, methods and attributes in the documentation.

2.7.4 Rule 23

Javadoc is not at all complete, but rather completely empty other than a one-line description of the class, hence the rule is definitely not abided by.

The class should be adequately explained in the documentation, both in its entirety and for each attribute and method.

2.8 Package and Import Statements

2.8.1 Rule 24

The only package statement is also the first non-comment statement, and import statements follow it, hence the rule is correctly followed.

2.9 Class and Interface Declarations

2.9.1 Rule 25

The class declarations are in the correct order, so the rule can be deemed as respected.

2.9.2 Rule 26

Methods are reasonably grouped by functionality, hence the rule is abided by.

2.9.3 Rule 27

The method *visitModelForm* at L. 542 is 104 lines of code long and could therefore be split in two or more methods for better readability; the same applies for the method *visit(ModelMenuItem modelMenuItem)* which is 59 lines of code long.

The size of the other methods can be instead deemed appropriate for their purpose.

No method or attribute duplicates are found in the class, and the class size can be considered appropriate for the functionalities it purports to provide.

Alas, coupling and cohesion can be considered adequate.

2.10 Initialization and Declarations

2.10.1 Rule 28

All variables and class members are of the correct type. They have the right visibility (public/private/protected).

2.10.2 Rule 29

Variables are declared in the proper scope.

2.10.3 Rule 30

Constructors are called when a new object is desired.

2.10.4 Rule 31

All object references are initialized before use.

2.10.5 Rule 32

Variables are declared at the beginning of the class and initialized in the constructor.

2.10.6 Rule 33

Declarations appear at the beginning of blocks (A block is any code surrounded by curly braces '{' and '}').

2.11 Method Calls

2.11.1 Rule 34

Parameters are presented in the correct order.

2.11.2 Rule 35

The correct methods are being called.

2.11.3 Rule 36

Method returned values are used properly.

2.12 Arrays

2.12.1 Rule 37

No arrays are used so there are no off-by-one errors in array indexing.

2.12.2 Rule 38

No collections are accessed via a direct index so all array (or other collection) indexes have been prevented from going out-of-bounds.

2.12.3 Rule 39

No arrays are used so no constructors are called when a new array item is desired.

2.13 Object Comparison

2.13.1 Rule 40

No objects are compared with ==, hence the rule is abided by.

2.14 Output Format

2.14.1 Rule 41

No output is ever displayed, so the rule is correctly followed.

2.14.2 Rule 42

No output is ever displayed; to make the class easier to interface with, some error messages should be given to whoever uses this class in any other part of the project, e.g. when dealing with exceptions.

2.14.3 Rule 43

No output is ever displayed, so the rule is correctly followed.

2.15 Computation, Comparisons and Assignments

2.15.1 Rule 44

No statement makes use of *brutish programming*.

2.15.2 Rule 45

Order of computation/evaluation, operator precedence and parenthesizing are correct.

2.15.3 Rule 46

As a corollary of previous rule, parenthesis are correctly used to avoid precedence problems.

2.15.4 Rule 47

No denominators of a division could have zero value.

2.15.5 Rule 48

No truncation/rounding problem detected as a result of an integer arithmetic operation.

2.15.6 Rule 49

Comparison and Boolean operators are correct.

2.15.7 Rule 50

Error conditions of throw-catch blocks are legitimate.

2.15.8 Rule 51

The code is free of any implicit type conversion.

2.16 Exceptions

2.16.1 Rule 52

All the relevant exceptions are caught. In fact, every method in the class re-throws any kind of Exception.

2.16.2 Rule 53

The code has no occurrence of catch blocks.

2.17 Flow of Control

2.17.1 Rule 54

There are no switch statements, therefore the rule is respected.

2.17.2 Rule 55

There are no switch statements, therefore the rule is respected.

2.17.3 Rule 56

All loops are correctly formed, with the appropriate initialization, increment and termination expressions (actually, all the used loops are for-each ones), therefore the rule is correctly followed.

2.18 Files

2.18.1 Rule 57

No files are opened, hence the rule is abided by.

2.18.2 Rule 58

No files are opened, hence the rule is abided by.

2.18.3 Rule 59

No files are opened, hence the rule is abided by.

2.18.4 Rule 60

No files are opened, hence the rule is abided by.

3 Other Issues

Here follows a list of other issues identified outside of the inspection of the rules provided in the checklist:

- L. 342 is a useless empty row (just a semicolon)
- L. 330, 359 contain chained calls to two methods with the same name but from different classes (respectively *ModelTreeNode* and *ModelTreeCondition*), making the code not easily readable.

4 Effort Spent

Arcari Leonardo

- 03/02/2017 - 2h

Bertoglio Riccardo

- 28/01/2017 - 45min
- 03/02/2017 - 1h
- 04/02/2017 - 1h

Galimberti Andrea

- 21/01 - 1h
- 26/01 - 1h
- 27/01 - 1h
- 31/01 - 2h
- 02/02 - 2h
- 03/02 - 1h