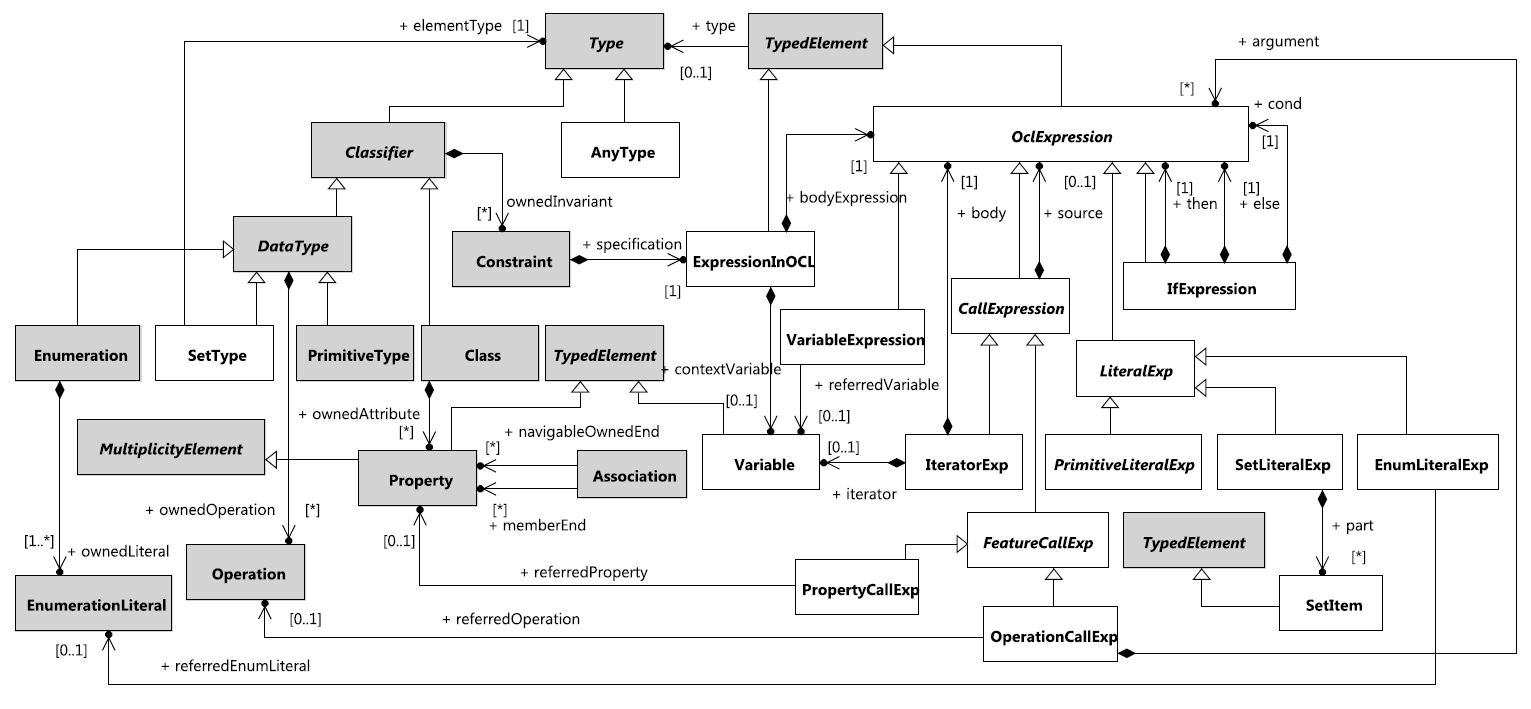
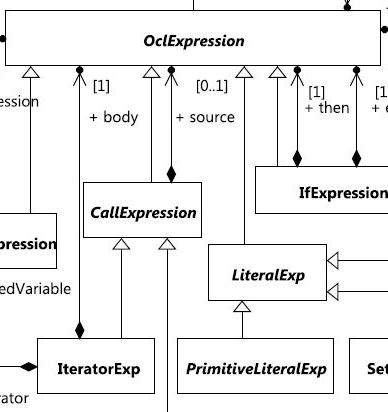
**Bericht/Erläuterung zur Implementierung der OCL Smells**

**Aufbau der OCL:**

****

Dieses Schaubild zeigt die abstrakte Syntax der OCL. Dabei stammen die grau hinterlegten Elemente aus der UML, die Elemente mit weißem Hintergrund aus der OCL. Die OCL ist eine streng getypte Sprache, was daran erkennbar ist, dass jedes Element der OCL von "Type" oder "TypedElement" (über "OclExpression") erbt.

**Magic Literal:**

****

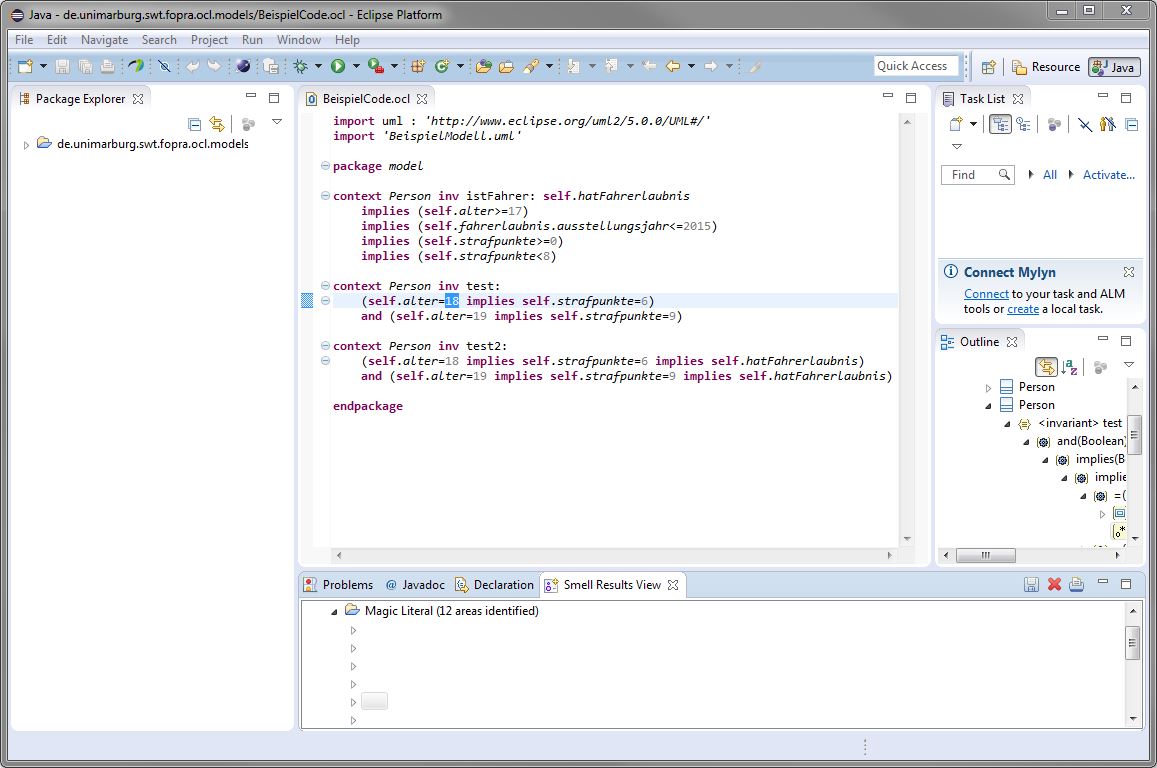
Der "Magic Literal" Smell beschreibt Werte im OCL-Code, die nicht an Variablen gebunden sind. Werte sollten an Variablen gebunden sein, da der entsprechende Wert im Falle einer Änderung an jeder Stelle, an der er verwendet wird, angepasst werden muss. In dem Bildausschnitt zum Aufbau der OCL kann man erkennen, dass dafür nach einer "LiteralExpression", genauer nach einer "PrimitiveLiteralExpression" gesucht werden muss.

**Meine Implementierung dieses Smells:**

1. Root oclRoot = OCLUtil.*getOCLPivotRoot*(root);
2. **if** (oclRoot != **null**) {
3. EList<EObject> allOCLElements = OCLUtil.*getAllOCLElementsFromRoot*(oclRoot);
4. **for** (EObject elem : allOCLElements) {
5. **if** (elem **instanceof** PrimitiveLiteralExp) {
6. OCLUtil.*addModelSmell*(results, elem, root);
7. }
8. }
9. }

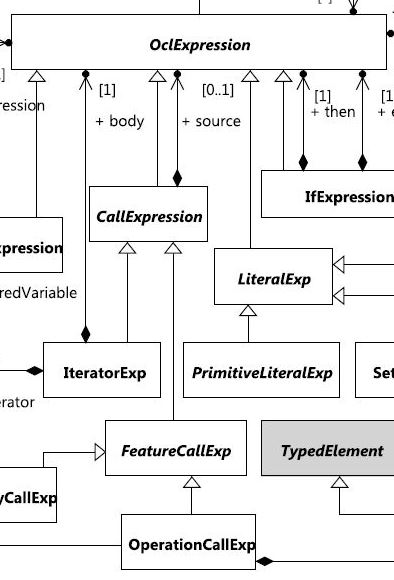
Die Idee dieser Implementierung ist es, der Reihe nach alle Elemente des vorliegenden OCL-Codes zu untersuchen und darin vorkommende, nicht an Variablen gebundene Werte zu finden. Dies wird erreicht, indem eine Liste aller Elemente in diesem Code erstellt wird (Z. 3). Danach wird über alle Elemente dieser Liste iteriert (ab Z. 4). Dabei wird bei jedem Element überprüft, ob dieses vom Typ "PrimitiveLiteralExp " ist (Z. 5), wobei es sich um eine "PrimitiveLiteralExpression" handelt. Dies ist der allgemeine Typ eines Wertes, der an keine Variable gebunden ist. Wird ein Element dieses Typs gefunden, wird es in die Liste "results" eingefügt (Z. 6), welche alle gefundenen Vorkommen des "Magic Literal" Smells beinhalten wird und als Ergebnis der Methode dient.

**Beispiel der Anwendung:**

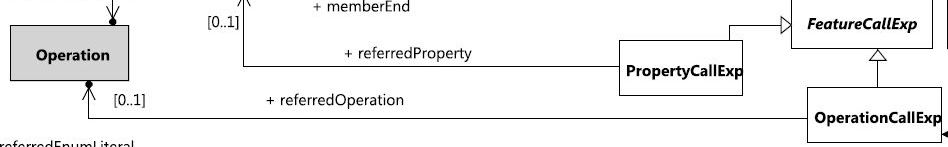
****

Wie man oben sehen kann, wurden in diesem Beispiel 12 Vorkommen des "Magic Literal" Smells gefunden. Der jeweilige Smell wird im Code markiert, wenn er in der "Smell Results View" ausgewählt wird. In diesem Beispiel ist das der Wert 18, also der fünfte gefundene "Magic Literal" Smell. Die entsprechenden Elemente werden in der "Smell Results View" nicht richtig angezeigt, da es hier zu einem Casting-Fehler (BigInteger zu String) kommt.

**Implies Chain:**



Der "Implies Chain" Smell beschreibt Verkettungen von mehreren "implies"-Operationen, wobei diese aufeinander folgen müssen und nicht durch andere Operationen wie bspw. "and" getrennt sein dürfen. Wie der Bildausschnitt zeigt, muss dafür nach einer "CallExpression" gesucht werden. Verfolgt man weiter den Aufbau der OCL, kann man erkennen, dass es sich bei dieser "CallExpression" um eine "FeatureCallExpression", genauer eine "OperationCallExpression" handeln muss. Über "referredOperation" erreicht man von dort die entsprechende Operation.

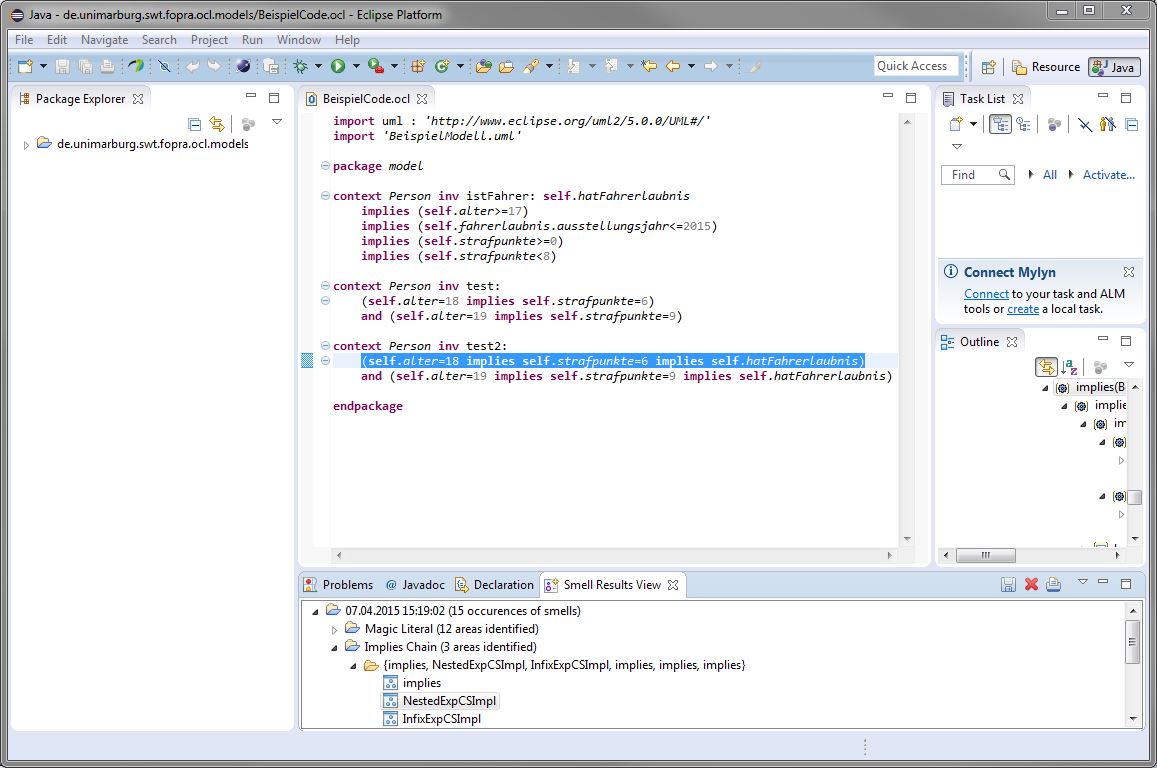


**Meine Implementierung dieses Smells:**

1. Root oclRoot = OCLUtil.*getOCLPivotRoot*(root);
2. **if** (oclRoot != **null**) {
3. EList<EObject> allOCLElements = OCLUtil.*getAllOCLElementsFromRoot*(oclRoot);
4. **for** (EObject elem : allOCLElements) {
5. **if** (elem **instanceof** ExpressionInOCL) {
6. ExpressionInOCL inv = (ExpressionInOCL) elem;
7. EList<EObject> usages = **new** BasicEList<EObject>();
8. EList<EObject> allInvariantElements =
9. OCLUtil.*getAllOCLElementsFromInvariant*(inv);
10. **for** (EObject invElem : allInvariantElements) {
11. **if** (invElem **instanceof** OperationCallExp) {
12. OperationCallExp OpCExp = (OperationCallExp) invElem;
13. **if** (OpCExp.getReferredOperation().getName().equals("implies")) {
14. **if** (OpCExp.getSource() **instanceof** OperationCallExp) {
15. OperationCallExp OpCExpSource = (OperationCallExp) OpCExp.getSource();
16. **if** (OpCExpSource.getReferredOperation().getName().equals("implies")) {
17. **if**(!usages.contains(OpCExp)){
18. usages.add(OpCExp);
19. }
20. usages.add(OpCExpSource);
21. }
22. **else** {
23. **if** (usages.size() > 1) {
24. OCLUtil.*addModelSmell*(results, usages, root);
25. }
26. usages.clear();
27. }
28. }
29. **else** {
30. **if** (usages.size() > 1) {
31. OCLUtil.*addModelSmell*(results, usages, root);
32. }
33. usages.clear();
34. }
35. }
36. }
37. }
38. }
39. }
40. }

Auch in dieser Implementierung werden der Reihe nach alle Elemente des OCL-Codes nach den richtigen Elementen, hier "implies"-Operationen, durchsucht. Der Unterschied hier ist allerdings, dass es für die richtige Erkennung des "Implies Chain" Smells nicht ausreicht, nur alle Vorkommen von "implies" zu finden. Da "Implies Chain" nur durch die Verkettung mehrerer "implies"-Operationen hintereinander in einer Invariante entsteht, muss jeweils auch das Source-Element der gefundenen Operationen überprüft werden. Dazu wird erst eine Liste mit allen Elementen des Codes erstellt (Z. 3). Danach werden aus dieser Liste neue Listen erstellt, die jeweils alle Elemente einer einzelnen Invariante enthalten (Z.5-8). So wird sichergestellt, dass der Smell wirklich in einer Invariante auftritt und sich nicht über mehrere Invarianten erstreckt. Um jetzt alle "implies"-Operationen in einer Invariante zu finden, wird über alle Elemente der Liste iteriert und überprüft, ob es sich bei dem aktuellen Element um eine Operation, also eine "OperationCallExp" handelt (Z. 10). Ist dies der Fall, wird abgefragt, ob diese Operation "implies" ist (Z. 12-13). Sofern diese Bedingung auch erfüllt ist, wird das Source-Element dieser Operation auf die gleiche Art überprüft (Z. 14-16). Erst wenn auch dieses Element eine "implies"-Operation ist, ist ein "Implies Chain" Smell gefunden, und beide Elemente werden in die Liste "usages" eingefügt (Z. 17-20). Dabei wird überprüft, ob das erste Element möglicherweise bereits in der Liste enthalten ist (Z. 17), da durch die oben beschriebene Überprüfung des Source-Elements immer zwei Elemente in die Liste eingefügt werden. Ist das Source-Element keine "implies"-Operation, wird der aktuelle Smell als vollständig gefunden angesehen und in die Ergebnisliste "results" eingefügt sowie die Liste "usages" geleert (Z. 22-26). Diese Überprüfung findet danach auch noch für Elemente, die normal über den Iterator aufgerufen werden, statt (Z.29-33). Wenn alle Vorkommen des Smells in einer Invariante gefunden sind, beginnt die Untersuchung der nächsten Invariante. Durch die Untersuchung aller Invarianten nach diesem Prinzip werden alle Vorkommen des "Implies Chain" Smells im OCL-Code gefunden.

**Beispiel der Anwendung:**



Oben im Bild kann man erkennen, dass im gewählten Beispielcode nur ein "Implies Chain" Smell gefunden wurde. Nur die Invariante "istFahrer" enthält drei gültige "Implies Chain" Smells, da dort die "implies"-Operationen direkt aufeinander folgen. In der Invariante "test" wird demnach kein "Implies Chain" Smell gefunden, da die beiden "implies"-Operationen dort nicht aufeinander folgen, sondern von einer "and"-Operation getrennt sind.

In der Invariante "test2" werden zwei Vorkommen des Smells gefunden, die durch ein "and" getrennt werden. In diesem Beispiel wird der erste Smell in "test2" ausgewählt und damit im Code markiert.