Frederik Alexander Hounsvad – frhou18@student.sdu.dk – Assignment 2: Math Interpreter

All tests were passed, validation implemented, scoping implemented, generator implemented, hoverbox not implemented.

```
grammar dk.sdu.mmmi.mdsd.Math with org.eclipse.xtext.common.Terminals

generate math "http://www.sdu.dk/mmmi/mdsd/Math"

Variables:
    variableAssignments+=VariableAssignment+
;

VariableAssignment returns Variable: //Serves as a basis to retain results and to be the basis for lines in the dsl
    {VariableAssignment} 'var' name=ID '=' exp=Exp
;

Exp returns Expression: //Addition and subtraction - Can boil down to MultDiv
    MultDiv (('+' {Plus.left=current}| '-' {Minus.left=current}) right=MultDiv)*
;

MultDiv returns Expression: //Multiplication and devision - Can boil down to MultDiv
    Primary (('*' {Multiplication.left=current}| '/' {Division.left=current}) right=Primary)*
;

Primary returns Expression: //Numbers and things that should be computed down to numbers before use
    Number | Parenthesis | VariableUse | LocalAssignment
;

Parenthesis returns Expression: //Serves to support the use of parentheses as a base
    {Parenthesis} '(' exp=Exp ')'
;

Number returns Expression: //A basic number
    {ExplicitNumber} value=INT
;

VariableUse: //Using a previously defined variable
    {VarUse} ref=[Variable]
;

Assignment returns Variable:
    {Assignment} name=ID '=' exp=Exp
;

LocalAssignment: //This is kind of like a using statement, where an alias is made for an expression or simmilar that only exists in the body of the let
```

```
  statement
42    {Local} 'let' assignment=Assignment 'in' exp=Exp 'end'
43;
44
45
```

```
 2  * generated by Xtext 2.25.0
 4 package dk.sdu.mmmi.mdsd.generator
 5
 6 import dk.sdu.mmmi.mdsd.math.Division
25
26 /**
27  * Generates code from your model files on save.
28  *
29  * See https://www.eclipse.org/Xtext/documentation/
   303_runtime_concepts.html#code-generation
30  */
31 class MathGenerator extends AbstractGenerator {
32
33     static Map<String, Integer> variables = new HashMap();
34
35     override void doGenerate(Resource resource, IFileSystemAccess2 fsa,
   IGeneratorContext context) {
36         val lines = resource.allContents.filter(Variables).next
37         val result = lines.compute
38
39         result.displayPanel
40     }
41
42     def static Map<String, Integer> compute(Variables math){
43         var values = new HashMap<String, Integer>
44         for (varass : math.getVariableAssignments()) {
45             values.put(varass.getName(), ComputeExp(varass))
46         }
47         return values
48     }
49
50     //Plus
51     def static dispatch Integer ComputeExp(Plus exp) {
52         return exp.left.ComputeExp() + exp.right.ComputeExp()
53     }
54     //Minus
55     def static dispatch Integer ComputeExp(Minus exp) {
56         return exp.left.ComputeExp() - exp.right.ComputeExp()
57     }
58     //Multiplication
59     def static dispatch Integer ComputeExp(Multiplication exp) {
60         return exp.left.ComputeExp() * exp.right.ComputeExp()
61     }
62     //Division
63     def static dispatch Integer ComputeExp(Division exp) {
64         return exp.left.ComputeExp() / exp.right.ComputeExp()
65     }
66     //ExplicitNumber
```

```
67     def static dispatch Integer ComputeExp(ExplicitNumber exp) {
68         return exp.value
69     }
70     //Parenthesis
71     def static dispatch Integer ComputeExp(Parenthesis exp) {
72         return exp.getExp.ComputeExp()
73     }
74     //VarUse
75     def static dispatch Integer ComputeExp(VarUse exp) {
76         return exp.ref.ComputeExp()
77     }
78     //Let
79     def static dispatch Integer ComputeExp(Local exp) { //Let
80         return exp.exp.ComputeExp()
81     }
82     //Variable
83     def static dispatch Integer ComputeExp(Variable exp) {
84         return exp.exp.ComputeExp()
85     }
86 
87     def void displayPanel(Map<String, Integer> result) {
88         var resultString = ""
89         for (entry : result.entrySet()) {
90             resultString += "var " + entry.getKey() + " = " + entry.getValue()
   + "\n"
91         }
92 
93         JOptionPane.showMessageDialog(null, resultString ,"Math Language",
   JOptionPane.INFORMATION_MESSAGE)
94     }
95 }
96 
```

```
 2 * generated by Xtext 2.26.0
 4 package dk.sdu.mmmi.mdsd.scoping
 5
 6 import dk.sdu.mmmi.mdsd.math.Assignment
 7 import dk.sdu.mmmi.mdsd.math.Local
 8 import dk.sdu.mmmi.mdsd.math.VarUse
 9 import dk.sdu.mmmi.mdsd.math.Variable
10 import dk.sdu.mmmi.mdsd.math.VariableAssignment
11 import java.util.ArrayList
12 import java.util.List
13 import org.eclipse.emf.ecore.EObject
14 import org.eclipse.emf.ecore.EReference
15 import org.eclipse.xtext.EcoreUtil2
16 import org.eclipse.xtext.scoping.IScope
17 import org.eclipse.xtext.scoping.Scopes
18
19 /**
20  * This class contains custom scoping description.
21  *
22  * See https://www.eclipse.org/Xtext/documentation/
   303_runtime_concepts.html#scoping
23  * on how and when to use it.
24  */
25 class MathScopeProvider extends AbstractMathScopeProvider {
26
27     override IScope getScope(EObject context, EReference reference){
28         var scope = super.getScope(context, reference)
29         if(context instanceof VarUse){
30
31             var IScope returnScope = null
32
33             var letDefinition = EcoreUtil2.getContainerOfType(context, Local)
34             var letVariable = EcoreUtil2.getContainerOfType(context,
   Assignment)
35             if(letDefinition !== null && letVariable !==
   letDefinition.assignment){
36                 returnScope = addLetDefinition(letDefinition, context)
37             }else{
38                 if(letDefinition !== null){
39                     letDefinition =
   EcoreUtil2.getContainerOfType(letDefinition.eContainer, Local)
40                 }
41                 if(letDefinition !== null){
42                     returnScope = addLetDefinition(letDefinition, context)
43                 }else{
44                     returnScope = getVariableAssignmentsInScope(context);
45                 }
46             }
```

```
47              return returnScope
48          }
49          return scope;
50      }
51      protected def IScope addLetDefinition(Local letDefinition, EObject
  context){
52          val containingLet =
  EcoreUtil2.getContainerOfType(letDefinition.eContainer, Local)
53
54          if(containingLet === null){
55              return Scopes.scopeFor(#[letDefinition.assignment],
  getVariableAssignmentsInScope(context))
56          }else{
57              return Scopes.scopeFor(#[letDefinition.assignment],
  addLetDefinition(containingLet, context))
58          }
59      }
60
61      protected def IScope getVariableAssignmentsInScope(EObject context){
62          val root = EcoreUtil2.getRootContainer(context);
63          val List<EObject> candidates = new ArrayList();
64          //Get all variableAssignments
65          for(VariableAssignment assignment:
  EcoreUtil2.getAllContentsOfType(root, VariableAssignment)){
66              candidates.add(assignment as EObject)
67          }
68
69          //Should generate a list of all variables that are not let (ie the
  var difinitions that are global)
70          val List<EObject> variableAssignments = candidates
71          .filter(variable | variable !== EcoreUtil2.getContainerOfType(context,
  Variable))
72          .toList()
73
74          return Scopes.scopeFor(variableAssignments)
75      }
76  }
77
```

```
 2 * generated by Xtext 2.26.0
 4 package dk.sdu.mmmi.mdsd.validation
 5
 6 import dk.sdu.mmmi.mdsd.math.*
 7 import org.eclipse.xtext.EcoreUtil2
 8 import org.eclipse.xtext.validation.Check
 9
10 /**
11  * This class contains custom validation rules.
12  *
13  * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#validation
14  */
15 class MathValidator extends AbstractMathValidator {
16
17 //  public static val INVALID_NAME = 'invalidName'
18 //
19 //  @Check
20 //  def checkGreetingStartsWithCapital(Greeting greeting) {
21 //      if (!Character.isUpperCase(greeting.name.charAt(0))) {
22 //          warning('Name should start with a capital',
23 //                  MathPackage.Literals.GREETING__NAME,
24 //                  INVALID_NAME)
25 //      }
26 //  }
27
28     public static val DUPLICATE_NAME = 'duplicateName'
29
30     @Check
31     def GlobalVarDuplicate(VariableAssignment varAss){
32         var base = EcoreUtil2.getContainerOfType(varAss, Variables)
33         if(base.variableAssignments.filter[it !== varAss && it.name == varAss.name ].toList.size > 0){
34             println("Should have err")
35             error('Global variables cannot be assigned with the same name', MathPackage.Literals.VARIABLE__NAME, DUPLICATE_NAME)
36         }
37     }
38
39 }
40
```