Assignment 2 | MDSD

I've implemented the calculation logic, with correct handling of PEMDAS, added validation for reasigning "var"s and added scopping/shadowing. I've gotten all tests to pass. I've not implemented hovering.

Xtext

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

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

Math:
    lines += MathExp+
;

MathExp:
    'var' value=Assignment
;

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

Exp returns Expression:
    SubAddExp
;
//Reverse PMDAS
SubAddExp returns Expression:
    DivMultExp (('-' {Subtraction .left=current} | '+'
{Addition.left=current})  right=DivMultExp)*
;


DivMultExp returns Expression:
    Primary (('/' {Division.left=current} | '*'
{Multiplication.left=current}) right=Primary)*
;

Primary returns Expression:
    Number | Parenthesis | VariableUse | Let
;

Parenthesis returns Expression:
    '(' Exp ')'
;

Number returns Expression:
    {Number} value=INT
;
```

```
VariableUse returns Expression:
    {VarUse} ref = [Assignment]
;

Let :
    {Let} 'let' value=Assignment 'in' exp=Exp 'end'
;
```

Generator

```
 * generated by Xtext 2.25.0
 */
package dk.sdu.mmmi.mdsd.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.AbstractGenerator
import org.eclipse.xtext.generator.IFileSystemAccess2
import org.eclipse.xtext.generator.IGeneratorContext
import dk.sdu.mmmi.mdsd.math.MathExp
import java.util.Map
import java.util.HashMap
import dk.sdu.mmmi.mdsd.math.*
import java.util.Collections
import java.util.List
import java.util.Comparator
import java.util.ArrayList
import javax.swing.JOptionPane

/**
 * Generates code from your model files on save.
 *
 * See
https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-
generation
 */
class MathGenerator extends AbstractGenerator {

    static Map<String, Integer> variables = new HashMap();
    static List<MathExp> linesToBeProcessed = new ArrayList();
    override void doGenerate(Resource resource, IFileSystemAccess2 fsa,
IGeneratorContext context) {
        val math = resource.allContents.filter(Math)
        if (math.hasNext){
            val mathObj = math.next
            mathObj.compute.displayPanel
        }
    }

    def static compute(Math math) {
        variables = new HashMap();
        linesToBeProcessed=new ArrayList(math.lines)
```

```
        math.lines.forEach[line|
                line.compute
        ]
        variables
    }

    def static void compute(MathExp math) {
        if (linesToBeProcessed.contains(math)){
            variables.put(math.value.name,
math.value.exp.computeExp(variables))
            linesToBeProcessed.remove(math)
        }
    }

    def static int computeExp(Expression exp, Map<String, Integer> vars) {
        switch exp {
            Subtraction: exp.left.computeExp(vars)-
exp.right.computeExp(vars)
            Addition: exp.left.computeExp(vars)+exp.right.computeExp(vars)
            Multiplication:
exp.left.computeExp(vars)*exp.right.computeExp(vars)
            Division: exp.left.computeExp(vars)/exp.right.computeExp(vars)
            Number: exp.value
            VarUse: {
                    if (exp.ref.name === null) {
                        throw new Exception("The calculation references a
variable, that points to null. This is likely a scoping issue")
                    }
                    if(vars.containsKey(exp.ref.name)){
                        vars.get(exp.ref.name)
                    } else {
                        for (MathExp line : new
ArrayList(linesToBeProcessed)){
                            if (line.value.name == exp.ref.name){
                                line.compute
                                if(variables.containsKey(exp.ref.name)){
                                    val calculatedValue =
variables.get(exp.ref.name)
                                    vars.put(exp.ref.name, calculatedValue)
                                    return calculatedValue
                                } else {
                                    throw new Exception("Adhoc calculation
of " + exp.ref.name + " did not add a value to the global variable")
                                }
                            }
                        }
                        throw new Exception("The calculation references a
variable, that is not in the list of MathExp")
                    }
                }
            Let: {
                val localVars = new HashMap(vars);
                localVars.put(exp.value.name,
exp.value.exp.computeExp(localVars))
```

```
                exp.exp.computeExp(localVars)
            }
            default: throw new Exception("Unhandled expression: " + exp)
        }
    }

    def void displayPanel(Map<String, Integer> result) {
        var resultString = ""
        for (entry : result.entrySet()) {
            resultString += "var " + entry.getKey() + " = " +
entry.getValue() + "\n"
        }
        println()

        JOptionPane.showMessageDialog(null, resultString ,"Math Language",
JOptionPane.INFORMATION_MESSAGE)
    }

}
```

Validator

```
 * generated by Xtext 2.25.0
 */
package dk.sdu.mmmi.mdsd.validation

import dk.sdu.mmmi.mdsd.math.*
import org.eclipse.xtext.validation.Check
import dk.sdu.mmmi.mdsd.math.MathPackage
import org.eclipse.xtext.EcoreUtil2

/**
 * This class contains custom validation rules.
 *
 * See
https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#valid
ation
 */
class MathValidator extends AbstractMathValidator {

    public static val DUPLICATE_NAME = 'duplicateName'

    @Check
    def cannotReasignGlobalVar(MathExp mathexp) {
        val root = EcoreUtil2.getContainerOfType(mathexp, Math)
        if(root.lines.filter[it !== mathexp && it.value.name ==
mathexp.value.name].size > 0) {
            error('Cannot assign global variable with same name',
                    MathPackage.Literals.MATH_EXP__VALUE,
                    DUPLICATE_NAME)
        }
```

```
    }

}
```

Scoping Provider

```
 * generated by Xtext 2.25.0
 */
package dk.sdu.mmmi.mdsd.scoping

import org.eclipse.emf.ecore.EObject
import org.eclipse.emf.ecore.EReference
import dk.sdu.mmmi.mdsd.math.*
import org.eclipse.xtext.scoping.IScope
import org.eclipse.xtext.scoping.Scopes
import org.eclipse.xtext.EcoreUtil2
import dk.sdu.mmmi.mdsd.math.Let
import java.util.List
import dk.sdu.mmmi.mdsd.math.Assignment

/**
 * This class contains custom scoping description.
 *
 * See
https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#scopi
ng
 * on how and when to use it.
 */
class MathScopeProvider extends AbstractMathScopeProvider {

        override getScope(EObject context, EReference reference) {
            var scope = super.getScope(context, reference)
            if (context instanceof VarUse){
                val rootElement = EcoreUtil2.getRootContainer(context);
                val List<Assignment> candidates =
EcoreUtil2.getAllContentsOfType(rootElement, Assignment)
                val List<Assignment> validCandidates = candidates.filter(a
| a.eContainer instanceof MathExp).filter(a | a.eContainer !==
EcoreUtil2.getContainerOfType(context, MathExp)).toList();

                val let = EcoreUtil2.getContainerOfType(context, Let)
                if (let !== null){
                    validCandidates.addLets(let, candidates)
                }

                return Scopes.scopeFor(validCandidates);
            }

            return scope;
        }
```

```
        def void addLets(List<Assignment> validCandidates, Let let,
List<Assignment> candidates){
            validCandidates.add(candidates.findFirst(a | a.eContainer ===
let))
            val newLet = EcoreUtil2.getContainerOfType(let.eContainer, Let)
            if (newLet !== null){
                validCandidates.addLets(newLet, candidates)
            }
        }
}
```