

# Assignment 02 — Nikolai Emil Damm

---

## My code

[GitHub — devantler/mdsd-assignment02](#)

## Assignment status

I have solved the assignment such that all tests pass. I have also tested that the solution works in practice.

I have not implemented hovering.

## My grammar language

```
Model:
    variables+=GlobalVariable*;

GlobalVariable returns Variable:
    {GlobalVariable}'var' name=ID '=' expression=AdditionExpression;

LocalVariable returns Variable:
    {LocalVariable}'let' name=ID '=' local_expression=AdditionExpression
    'in' expression=AdditionExpression 'end';

AdditionExpression returns Expression:
    SubtractionExpression ({AdditionExpression.left=current} '+'
    right=SubtractionExpression)*;

SubtractionExpression returns Expression:
    MultiplicationExpression ({SubtractionExpression.left=current} '-'
    right=MultiplicationExpression)*;

MultiplicationExpression returns Expression:
    DivisionExpression ({MultiplicationExpression.left=current} '*'
    right=DivisionExpression)*;

DivisionExpression returns Expression:
    ExpressionValue ({DivisionExpression.left=current} '/'
    right=ExpressionValue)*;

ExpressionValue returns Expression:
    ParenthesizedExpression | Number | LocalVariable | VariableReference;

ParenthesizedExpression returns Expression:
    '(' AdditionExpression ')';

Number:
    value=INT;
```

```
VariableReference:  
    variable=[Variable];
```

## My generator

```
class MathGenerator extends AbstractGenerator {  
  
    static Map<String, Integer> variables  
  
    override void doGenerate(Resource resource, IFileSystemAccess2 fsa,  
IGeneratorContext context) {  
        val model = resource.allContents.filter(Model).next  
        val result = model.compute  
  
        result.displayPanel  
    }  
  
    def static compute(Model model) {  
        variables = new HashMap()  
        for (variable : model.variables) {  
            val localVariables = new HashMap<String, Integer>();  
            variables.put(variable.name,  
variable.expression.computeExp(localVariables))  
        }  
        return variables  
    }  
  
    def dispatch static int computeExp(AdditionExpression expression,  
Map<String, Integer> localVariables) {  
        expression.left.computeExp(localVariables) +  
expression.right.computeExp(localVariables)  
    }  
  
    def dispatch static int computeExp(SubtractionExpression expression,  
Map<String, Integer> localVariables) {  
        expression.left.computeExp(localVariables) -  
expression.right.computeExp(localVariables)  
    }  
  
    def dispatch static int computeExp(MultiplicationExpression  
expression, Map<String, Integer> localVariables) {  
        expression.left.computeExp(localVariables) *  
expression.right.computeExp(localVariables)  
    }  
  
    def dispatch static int computeExp(DivisionExpression expression,  
Map<String, Integer> localVariables) {  
        expression.left.computeExp(localVariables) /  
expression.right.computeExp(localVariables)  
    }  
}
```

```
def dispatch static int computeExp(Number number, Map<String, Integer>
localVariables) {
    number.value
}

def dispatch static int computeExp(Variable variable, Map<String,
Integer> localVariables){
    val nestedVariables = new HashMap(localVariables);
    if(variable instanceof LocalVariable){
        nestedVariables.put(variable.name,
variable.local_expression.computeExp(nestedVariables))
    }
    variable.expression.computeExp(nestedVariables)
}

def dispatch static int computeExp(VariableReference reference,
Map<String, Integer> localVariables) {
    val globalVariable = variables.get(reference.variable.name)
    val localVariable = localVariables.get(reference.variable.name)
    if (reference.variable instanceof LocalVariable) {
        return localVariable != null ? localVariable : globalVariable
    } else {
        return globalVariable != null ? globalVariable :
reference.variable.computeExp(localVariables)
    }
}

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

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