Model-driven software development: Assignment 3

By Peter Brændgaard pebra18

Tests passed:

All

Link to repository

https://github.com/Peterzxcvbnm/my_assignment3

Math.xtext

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

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

MathExp:
	program=Program (externals+=External*)? variables += VarBinding*
;

External:
	'external' name=ID '(' (parameters+=ParmeterTypes ("," 
parameters+=ParmeterTypes)*)? ')'
;

enum ParmeterTypes:
	int='int' | string='string'
;

Program:
	'program' name=ID
;

VarBinding:
	'var' name=ID '=' expression=Exp
;

Exp returns Expression:
	Factor (( {Plus.left=current} '+' | {Minus.left=current}  '-' ) right=Factor)*
;

Factor returns Expression:
	Primary (({Mult.left=current} '*' | {Div.left=current}  '/'  ) right=Primary)*
;

Primary returns Expression:
	{MathNumber} value=INT  | Parenthesis  |  VariableUse | LetBinding | ExternalUse
;

ExternalUse:
	ref=[External] '(' (expressions+=Exp ("," expressions+=Exp)*)? ')'
```

```
;

LetBinding:
	'let' name=ID '=' binding=Exp 'in' body=Exp 'end'
;

Parenthesis:
	'(' expression=Exp ')'
;

Binding:
	VarBinding | LetBinding
;

VariableUse:
	ref = [Binding]
;
```

MathGenerator.xtend

```
* generated by Xtext 2.25.0

 */

package dk.sdu.mmmi.mdsd.generator


import dk.sdu.mmmi.mdsd.math.Div

import dk.sdu.mmmi.mdsd.math.LetBinding

import dk.sdu.mmmi.mdsd.math.MathExp

import dk.sdu.mmmi.mdsd.math.MathNumber

import dk.sdu.mmmi.mdsd.math.Minus

import dk.sdu.mmmi.mdsd.math.Mult

import dk.sdu.mmmi.mdsd.math.Plus

import dk.sdu.mmmi.mdsd.math.VarBinding

import dk.sdu.mmmi.mdsd.math.VariableUse

import java.util.HashMap

import java.util.Map

import javax.swing.JOptionPane

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.Program
import dk.sdu.mmmi.mdsd.math.Parenthesis
import dk.sdu.mmmi.mdsd.math.ExternalUse

/**
 * 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, String> variables;

	override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
		val className = resource.allContents.filter(Program).next
		val math = resource.allContents.filter(MathExp).next
		val variables = math.variables
		val result = math.compute

		var counter = 0;
		fsa.generateFile('''math_expression/«className.name».java''', '''
		package math_expression;
		public class «className.name» {

		«FOR variable : variables»
		public int «variable.name»;
		«ENDFOR»

		«IF math.externals.size > 0»
```

```
                    private External external;

                    public «className.name»(External external) {
                      this.external = external;
                    }
                    «ENDIF»

                    public void compute() {
                      «FOR variable : variables SEPARATOR '
                            '»
                            «variable.name» = «variable.computeExpression»;
«ENDFOR»
                    }

                  «IF math.externals.size > 0»
                    public interface External {
                            «FOR external : math.externals»
                      public int «external.name»(«FOR param : external.parameters SEPARATOR ', '»«param»
n«counter++»«ENDFOR»);
                      «ENDFOR»
                    }
                    «ENDIF»
                  }
                  ''')

        }

        def static compute(MathExp math) {
                  variables = new HashMap()
                  for(varBinding: math.variables)
```

```
                varBinding.computeExpression()
        variables
}

def static dispatch String computeExpression(VarBinding binding) {
        variables.put(binding.name, binding.expression.computeExpression())
        return variables.get(binding.name)
}

def static dispatch String computeExpression(MathNumber exp) {
        exp.value.toString
}

def static dispatch String computeExpression(Plus exp) {
        exp.left.computeExpression + " + " + exp.right.computeExpression
}

def static dispatch String computeExpression(Minus exp) {
        exp.left.computeExpression  + " - " + exp.right.computeExpression
}

def static dispatch String computeExpression(Mult exp) {
        exp.left.computeExpression  + " * " + exp.right.computeExpression
}

def static dispatch String computeExpression(Div exp) {
        exp.left.computeExpression  + " / " + exp.right.computeExpression
}

def static dispatch String computeExpression(LetBinding exp) {
```

```
		return "(" + exp.body.computeExpression + ")"
	}

	def static dispatch String computeExpression(VariableUse exp) {
		return "(" + exp.ref.computeExpression + ")"
	}

	def static dispatch String computeExpression(Parenthesis exp){
		return "( " + exp.expression.computeExpression + " )"
	}

	def static dispatch String computeExpression(ExternalUse exp){
		if(exp.expressions.size > 0){
			return '''this.external.«exp.ref.name»(«FOR expression : exp.expressions SEPARATOR ', '»«expression.computeExpression»«ENDFOR»)'''
		}else{
			return '''this.external.«exp.ref.name»()'''
		}
	}

	def static dispatch String computeBinding(VarBinding binding){
		if(!variables.containsKey(binding.name))
			binding.computeExpression()
		variables.get(binding.name)
	}

	def static dispatch String computeBinding(LetBinding binding){
		return "(" + binding.binding.computeExpression + ")"
	}
```

```
}
```