```
 2 * generated by Xtext 2.25.0
 4 package dk.sdu.mmmi.mdsd.generator
 5
 6 import dk.sdu.mmmi.mdsd.math.Division
30
31 /**
32  * Generates code from your model files on save.
33  *
34  * See https://www.eclipse.org/Xtext/documentation/
    303_runtime_concepts.html#code-generation
35  */
36 class MathGenerator extends AbstractGenerator {
37
38     static Map<String, Integer> variables = new HashMap();
39
40     override void doGenerate(Resource resource, IFileSystemAccess2 fsa,
    IGeneratorContext context) {
41         val program = resource.allContents.filter(Program).next
42         val className = program.programName.name
43         // Append Header to class
44         val math_expressionClass = getMathClass(program, className)
45         fsa.generateFile(className + ".java", math_expressionClass)
46     }
47
48     def getMathClass(Program program, String className) {
49         val contents = '''
50             «getHeader()»
51             package math_expression;
52             public class «className»{
53                 «getVariableDeclarations(program.variableAssignments)»
54
55                 «IF program.externals.length() > 0»
56                     private External external;
57
58                     public «className»(External external){
59                         this.external = external;
60                     }
61
62                     «getExternalInterface(program)»
63                 «ENDIF»
64
65                 public void compute(){
66                     «FOR instantiation : getVariableInstantiations(program)»
67                         «instantiation»;
68                     «ENDFOR»
69                 }
70
71             }
```

```
72          '''
73          //println(contents)
74          return contents
75      }
76
77      def getVariableInstantiations(Program program){
78          var vars = program.variableAssignments
79          return vars.compute()
80      }
81
82      def getExternalInterface(Program program) {
83          return '''
84              public interface External{
85                  «FOR external : program.externals»
86                      public int «getExternalSignature(external)»;
87                  «ENDFOR»
88              }
89          '''
90      }
91
92      def getExternalSignature(External external) {
93          var returnValue = external.name + "("
94          if (external.parameters.length == 1) {
95              returnValue += "int n"
96          } else if (external.parameters.length == 2) {
97
98              returnValue += "int n, int m"
99          }
100
101         returnValue += ")"
102         return returnValue
103     }
104
105     def getHeader() {
106         return '''
107             /*
108              * Generated by xtend by the generator made by 'Frederik
    Alexander Hounsvad' 'frhou18@student.sdu.dk'
109              * All Rights reserved
110             */
111         '''
112     }
113
114     def getVariableDeclarations(EList<Variable> list) {
115         return '''
116             «FOR variable : list»
117                 public int «variable.name»;
118             «ENDFOR»
```

```
119         '''
120     }
121 
122     def List<String> compute(EList<Variable> variables) {
123         var values = new ArrayList<String>();
124 
125         for (varass : variables) {
126             values.add(varass.name + " = " + ComputeExp(varass))
127         }
128         return values
129     }
130 
131     // Plus
132     def static dispatch String ComputeExp(Plus exp) {
133         return '''(«exp.left.ComputeExp()» + «exp.right.ComputeExp()»)'''
134     }
135 
136     // Minus
137     def static dispatch String ComputeExp(Minus exp) {
138         return '''(«exp.left.ComputeExp()» - «exp.right.ComputeExp()»)'''
139     }
140 
141     // Multiplication
142     def static dispatch String ComputeExp(Multiplication exp) {
143         return '''(«exp.left.ComputeExp()» * «exp.right.ComputeExp()»)'''
144     }
145 
146     // Division
147     def static dispatch String ComputeExp(Division exp) {
148         return '''(«exp.left.ComputeExp()» / «exp.right.ComputeExp()»)'''
149     }
150 
151     // ExplicitNumber
152     def static dispatch String ComputeExp(ExplicitNumber exp) {
153         return '''«exp.value»'''
154     }
155 
156     // Parenthesis
157     def static dispatch String ComputeExp(Parenthesis exp) {
158         return '''(«exp.getExp.ComputeExp()»)'''
159     }
160 
161     // VarUse
162     def static dispatch String ComputeExp(VarUse exp) {
163         return '''(«exp.ref.ComputeExp()»)'''
164     }
165 
166     // Let
```

```
167     def static dispatch String ComputeExp(Local exp) { // Let
168         return '''(«exp.exp.ComputeExp()»)'''
169     }
170 
171     // Variable
172     def static dispatch String ComputeExp(Variable exp) {
173         return '''(«exp.exp.ComputeExp()»)'''
174     }
175 
176     def static dispatch String ComputeExp(ExternalUse exp) {
177         var sb = new StringBuilder()
178         sb.append("(external.").append(exp.ref.name).append("(")
179         switch exp.ref.parameters.length(){
180             case 1: sb.append(ComputeExp(exp.exp.get(0)))
181             case 2:
    sb.append(ComputeExp(exp.exp.get(0))).append(",").append(ComputeExp(exp.exp.
    get(1)))
182         }
183         sb.append("))")
184         return sb.toString()
185     }
186 }
187 
```