1. Frederik Alexander Hounsvad, frhou18@student.sdu.dk:

2. Xtext Grammar:

   (a) Does your grammar support all of the example programs? If not what are the limitations?

   My grammar supports example programs 1 through 7.

   (b) How did you implement operator precedence and associativity?

   I went with the same strategy as in assignment 3, where every action is composed of a left and a right, with different types at the same precedence level. The left and the right are defined by the next type of value down the chan. And everything is optional until you hit the last level with primary values which has equal precedence level. Every rule is processed using this chaining, and things are wrapped in parentheses when useful.

   (c) How did you implement the syntax of variables (ID in rule Exp of the IF22 BNF), such that they can refer both to parameters of scenarios and local variables?

   I did not get to an example program which required this, and as such I have not implemented this functionality.

   (d) How did you implement the syntax of to statements, such that they refer to announcement/question/end/scenario? (first ID in rule Target of the IF22 BNF)

   I did not get to an example program which required this, and as such I have not implemented this functionality.

3. Scoping Rules

   (a) Did you implement scoping rules that allow variables to refer to variable definition and scenario parameters? If not, what are the limitations?

   I did not get to an example program which required this, and as such I have not implemented this functionality.

   (b) Did you implement scoping rules that allow announcement and question statements to call scenarios and reference the called scenario end statements? If not, what are the limitations?

   I did not get to an example program which required this, and as such I have not implemented this functionality.

   (c) Describe your implementation of any scoping rules included with your system.

I did not get to an example program which required this, and as such I have not implemented this functionality.

4. Type Inference

(a) What validation rule did you implement, if any?

I have implemented detection of duplicate function names, and a requirement for at least one end statement.

(b) Did you implement validation for the correct use of keywords such as this and Type keywords (number, text)? If not, what are the limitations? If yes, briefly describe your approach

I did not do any validation on this.

(c) Did you implement validation with type inferencing? If yes, does it correctly check the types for all expressions (including input validation and conditions) in the provided IF22 examples? If not, what are the problems?

I did not implement validation on this

(d) If you implemented validation with type inferencing, briefly describe your approach.

I did not

5. Generator

(a) Does your code generator correctly generate code for all of the examples provided, and are you confident that it will also work for “similar” programs? If not, then what limitations are there?

It generates code that is valid for example 1 through 7, and I am confident that it would be able to do so for any programs that are at the same level of complexity as example 1 through 7.

My solution is unable to handle multiple scenarios, ending targets and getting a response from a scenario.

(b) Briefly describe how your code generator works.

Generation is split into three parts.

- Generating the Game file
  - This handles generating the game file, and takes into account the need for the external declaration as well as taking from the constructor and passing it to the scenario.
- Generating the External file
  - This file is generated based on the used functions from the if file

- Generating the Scenarios
    - This generates the scenario a bit at a time.
      First it generates the more static parts of the program, such as imports. Then it adds the variables to the scenarios and adds the first interaction to the list. Then it runs through each statement, and handles it based on the type of statements. The question statements are handled in their own function, as the complexity became high enough that I needed to move something around. The questions are processed based on the as keyword and the in keyword.
      All expressions are computed using the overloaded computeExpr functions that exists for each type of expression. These functions return strings formatted as java code.

6. Implementation: include your xtext grammar file and all implemented Xtend files (scoping, type inference, generator) verbatim.

```
grammar dk.sdu.frhou18.mdsd.IF22 with org.eclipse.xtext.common.Terminals

generate iF22 "http://www.sdu.dk/frhou18/mdsd/IF22"

Model:
    storyName = Story functions+=Function* scenarios+=Scenario+
;

Function:
    'function' name=ID '(' (parameters+=TypeUsage (','
  parameters+=TypeUsage)*)? ')' ':' type = Type
;



Story:
    'story' name=ID
;

Scenario:
    'scenario' name=ID '{' variables+=VariableDef* statemens+=Statement* '}'
;

Statement:
    End | Announce | Question
;

End:
    'end' name=ID endMessage=LogicExp
;

Announce:
    'announce' name=ID text=LogicExp target=Target
;

Question:
    'question' name=ID text=LogicExp 'as' asValue=LogicExp ('in'
  inVar=VarUse)? targets+=Target+
;

VariableDef:
    'var' name=ID ':' type=Type
;

enum Type:
    boolean | text | number
;

```

```
47 Target:
48     'to' target=[TargetTarget] ('if' logic=LogicExp)?
49 ;
50
51 TargetTarget:
52     Scenario | Statement
53 ;
54
55 StringProducer:
56     TextExp
57 ;
58
59 This returns Expression:
60     {This} 'this'
61 ;
62
63 LogicExp returns Expression:
64     LogicAndOR (('==' {Equals.left=current}| '!=' {NotEquals.left=current}|
   '<' {Less.left=current}| '>' {Greater.left=current}|
   '<=' {LessOrEquals.left=current}| '>='{GreaterOrEquals.left=current})
   right=LogicAndOR)*
65 ;
66
67 LogicAndOR returns Expression:
68     MathExp (('&&' {And.left=current}|'||' {Or.left=current}) right=MathExp)*
69 ;
70
71 LogicNot returns Expression:
72     {LogicNot} '!' ref = Primary
73 ;
74
75 Parentheses returns Expression:
76     {Parentheses} '(' ref=LogicExp ')'
77 ;
78
79 Boolean returns Expression:
80      {Boolean} val=BooleanValue
81 ;
82
83 enum BooleanValue:
84     TRUE='true' | FALSE='false'
85 ;
86
87 //Potentially no parenthesese in maths, java is gut boi
88
89 MathExp returns Expression:
90     MultDivMathExp (('+' {Plus.left=current} | '-' {Minus.left=current})
   right = MultDivMathExp)*
```

```
 91 ;
 92
 93 MultDivMathExp returns Expression:
 94     TextExp (('*' {Multiplication.left=current}| '/' {Division.left=current})
    right=TextExp)*
 95 ;
 96
 97 MathNumberExp returns Expression:
 98     {MathNumberExp} value=INT
 99 ;
100
101 TextExp returns Expression:
102     Primary (('&' {TextExp.left=current}) right=Primary)*//{TextExp}
    stringValues += Primary ('&' stringValues+=Primary)*
103 ;
104
105 Primary returns Expression:
106     FunctionUsage | LogicNot | Boolean | Parentheses | This | MathNumberExp |
    TextLiteral | VarUse | TypeUsage
107 ;
108
109 FunctionUsage returns Expression:
110     {FunctionUsage} name=[Function] '(' exps+=LogicExp (',' exps+=LogicExp)*
    ')'
111 ;
112
113 TextLiteral returns Expression:
114     {TextLiteral} text=STRING
115 ;
116
117 VarUse returns Expression:
118     {VarUse} ref=[VariableDef]
119 ;
120
121 TypeUsage returns Expression:
122     {TypeUsage} type=Type
123 ;
124
```

```
 2  * generated by Xtext 2.26.0
 4 package dk.sdu.frhou18.mdsd.validation
 5
 6 import dk.sdu.frhou18.mdsd.iF22.Function
 7 import org.eclipse.xtext.validation.Check
 8 import dk.sdu.frhou18.mdsd.iF22.Model
 9 import org.eclipse.xtext.EcoreUtil2
10 import dk.sdu.frhou18.mdsd.iF22.IF22Package
11 import dk.sdu.frhou18.mdsd.iF22.Statement
12 import dk.sdu.frhou18.mdsd.iF22.Scenario
13 import dk.sdu.frhou18.mdsd.iF22.End
14
15 /**
16  * This class contains custom validation rules.
17  *
18  * See https://www.eclipse.org/Xtext/documentation/
   303_runtime_concepts.html#validation
19  */
20 class IF22Validator extends AbstractIF22Validator {
21
22 //  public static val INVALID_NAME = 'invalidName'
23 //
24 //  @Check
25 //  def checkGreetingStartsWithCapital(Greeting greeting) {
26 //      if (!Character.isUpperCase(greeting.name.charAt(0))) {
27 //          warning('Name should start with a capital',
28 //                  IF22Package.Literals.GREETING__NAME,
29 //                  INVALID_NAME)
30 //      }
31 //  }
32
33     public static val DUPLICATE_NAME = 'duplicateName'
34
35     @Check
36     def checkFunctionNameNotDuplicate(Function function){
37         var base = EcoreUtil2.getContainerOfType(function, Model)
38         if(base.functions.filter[f|f !== function && f.name ==
   function.name].toList.size > 0){
39             error('Functions are not allowed to have the same name',
   IF22Package.Literals.FUNCTION__NAME, DUPLICATE_NAME)
40         }
41     }
42
43     public static val AT_LEAST_ONE_END = 'atLeastOneEnd'
44
45     @Check
46     def atLeastOneEndStatement(Scenario sc){
47         if(sc.statemens.filter[st|st instanceof End].toList.size == 0){
```

```
48            error('A scenario should have at least one end statement',
  IF22Package.Literals.SCENARIO__STATEMENS, AT_LEAST_ONE_END)
49        }
50    }
51
52}
53
```

```
 2  * generated by Xtext 2.26.0
 4 package dk.sdu.frhou18.mdsd.generator
 5
 6 import org.eclipse.emf.ecore.resource.Resource
 7 import org.eclipse.xtext.generator.AbstractGenerator
 8 import org.eclipse.xtext.generator.IFileSystemAccess2
 9 import org.eclipse.xtext.generator.IGeneratorContext
10 import dk.sdu.frhou18.mdsd.iF22.Model
11 import dk.sdu.frhou18.mdsd.generator.subgenerators.GameGenerator
12 import dk.sdu.frhou18.mdsd.generator.subgenerators.ScenarioGenerator
13 import dk.sdu.frhou18.mdsd.generator.subgenerators.ExternalGenerator
14
15 /**
16  * Generates code from your model files on save.
17  *
18  * See https://www.eclipse.org/Xtext/documentation/
   303_runtime_concepts.html#code-generation
19  */
20 class IF22Generator extends AbstractGenerator {
21
22     String pkg = "interactive_fiction"
23     override void doGenerate(Resource resource, IFileSystemAccess2 fsa,
   IGeneratorContext context) {
24         val model = resource.allContents.filter(Model).next
25         val storyName = model.storyName.name
26         val scenarios = model.scenarios
27
28         GameGenerator.doGenerate(fsa, pkg, model, storyName, scenarios);
29         if(model.functions.length > 0)
30         ExternalGenerator.doGenerate(fsa,pkg,model,storyName,scenarios)
31         for(scenario : scenarios){
32             ScenarioGenerator.doGenerate(fsa, pkg, model, storyName, scenario)
33         }
34     }
35     def public static String snakeCase(String input){
36         return input.split("(?=\\p{Upper})").join('_').toLowerCase
37     }
38 }
39
```

```
 1 package dk.sdu.frhou18.mdsd.generator.subgenerators
 2
 3 import org.eclipse.xtext.generator.IFileSystemAccess2
 8
 9 class GameGenerator {
10     def static void doGenerate(IFileSystemAccess2 fsa, String pkg, Model
   model, String storyName, EList<Scenario> scenarios){
11         var hasFunc = model.functions.length > 0;
12
13         fsa.generateFile('''«pkg»/«IF22Generator.snakeCase(storyName)»/
   Game.java''',
14             '''
15                 package «pkg».«IF22Generator.snakeCase(storyName)»;
16
17                 import java.io.IOException;
18                 import interactive_fiction.common.*;
19
20                 public class Game{
21                     public Scenario start;
22
23                     «IF hasFunc»External external«ENDIF»;
24                     public Game(«IF hasFunc»External external«ENDIF»){
25                         this.start = new Scenario«scenarios.get(0).name»(«IF
   hasFunc»external«ENDIF»);
26                     }
27
28                     public void play() throws IOException {
29                         start.interact();
30                     }
31                 }
32             '''
33         )
34     }
35 }
```

```
 1 package dk.sdu.frhou18.mdsd.generator.subgenerators
 2
 3 import org.eclipse.xtext.generator.IFileSystemAccess2
 4 import dk.sdu.frhou18.mdsd.iF22.Model
 5 import dk.sdu.frhou18.mdsd.iF22.Scenario
 6 import dk.sdu.frhou18.mdsd.iF22.End
 7 import dk.sdu.frhou18.mdsd.generator.IF22Generator;
 8 import dk.sdu.frhou18.mdsd.iF22.TextExp
 9 import dk.sdu.frhou18.mdsd.iF22.Announce
10 import dk.sdu.frhou18.mdsd.iF22.Question
11 import dk.sdu.frhou18.mdsd.iF22.TextLiteral
12 import dk.sdu.frhou18.mdsd.iF22.Type
13 import dk.sdu.frhou18.mdsd.iF22.Equals
14 import dk.sdu.frhou18.mdsd.iF22.Less
15 import dk.sdu.frhou18.mdsd.iF22.NotEquals
16 import dk.sdu.frhou18.mdsd.iF22.Greater
17 import dk.sdu.frhou18.mdsd.iF22.LessOrEquals
18 import dk.sdu.frhou18.mdsd.iF22.GreaterOrEquals
19 import dk.sdu.frhou18.mdsd.iF22.And
20 import dk.sdu.frhou18.mdsd.iF22.Or
21 import dk.sdu.frhou18.mdsd.iF22.LogicNot
22 import dk.sdu.frhou18.mdsd.iF22.Parentheses
23 import dk.sdu.frhou18.mdsd.iF22.Plus
24 import dk.sdu.frhou18.mdsd.iF22.Minus
25 import dk.sdu.frhou18.mdsd.iF22.Multiplication
26 import dk.sdu.frhou18.mdsd.iF22.Division
27 import dk.sdu.frhou18.mdsd.iF22.MathNumberExp
28 import dk.sdu.frhou18.mdsd.iF22.This
29 import org.eclipse.xtext.EcoreUtil2
30 import dk.sdu.frhou18.mdsd.iF22.VarUse
31 import dk.sdu.frhou18.mdsd.iF22.TypeUsage
32 import dk.sdu.frhou18.mdsd.iF22.VariableDef
33 import dk.sdu.frhou18.mdsd.iF22.Function
34 import dk.sdu.frhou18.mdsd.iF22.FunctionUsage
35
36 class ScenarioGenerator {
37
38     def static void doGenerate(IFileSystemAccess2 fsa, String pkg, Model
   model, String storyName, Scenario scenario) {
39         var hasFunc = model.functions.length > 0;
40         if(scenario.statemens.length == 0) return
41         var fileContents =
42         '''
43             package «pkg».«IF22Generator.snakeCase(storyName)»;
44
45             import java.io.IOException;
46             import interactive_fiction.common.*;
47
```

```
48              class Scenario«scenario.name» extends Scenario {
49                  «FOR variable : scenario.getVariables»
50                      «variable.type.correspondingJavaType» «variable.name»;
51                  «ENDFOR»
52                  «IF hasFunc»
53                  External external;
54                  public Scenario«scenario.name»(External external){
55                      this.external = external;
56                  }
57                  «ENDIF»
58
59                  public String interact() throws IOException {
60                      nextInteraction = "«scenario.statemens.get(0).name»";
61                      while(true){
62                          switch(nextInteraction){
63
64          '''
65          for (statement : scenario.statemens) {
66              if (statement instanceof End) {
67                  fileContents +=
68          '''
69
70                      case "«statement.name»":
71                          System.out.println(«(statement as
  End).endMessage.computeExpr»);
72                          return "«(statement as End).name»";
73          '''
74              } else if (statement instanceof Announce) {
75                  val announce = (statement as Announce)
76                  fileContents +=
77          '''
78
79                      case "«statement.name»":
80                          System.out.println(«announce.text.computeExpr»);
81                          nextInteraction="«announce.target.target.name»";
82                          break;
83          '''
84              } else if (statement instanceof Question) {
85                  fileContents = fileContents.handleQuestion(statement as
  Question);
86              }
87          }
88          fileContents +=
89          '''
90
91                      }
92                  }
93              }
```

```
 94            }
 95        '''
 96
 97        fsa.generateFile('''«pkg»/«IF22Generator.snakeCase(storyName)»/
   Scenario«scenario.name».java''', fileContents)
 98    }
 99
100    def static String handleQuestion(String fc, Question question) {
101        var fileContents = fc;
102        fileContents +=
103        '''
104
105                case "«question.name»":
106                    System.out.println(«question.text.computeExpr»);
107        '''
108        var hasTryCatch = false;
109        if(question.inVar === null && question.asValue instanceof TypeUsage){
110            switch ((question.asValue as TypeUsage).type) {
111                case Type.BOOLEAN: {
112                    hasTryCatch = true
113                    fileContents +=
114        '''
115
116                        try{
117                            boolean __«question.name» =
   Boolean.parseBoolean(br.readLine());
118        '''
119                }
120                case Type.NUMBER: {
121                    hasTryCatch = true
122                    fileContents +=
123        '''
124
125                        try{
126                            int __«question.name» =
   Integer.parseInt(br.readLine());
127        '''
128                }
129                case Type.TEXT: {
130                    fileContents +=
131        '''
132
133                        String __«question.name» = br.readLine();
134        '''
135                }
136            }
137        }else if(question.inVar === null && !( question.asValue instanceof
   TypeUsage ) && !(question.asValue instanceof FunctionUsage)){
```

```
138             val gottenType =
    EcoreUtil2.getAllContentsOfType(question.asValue, TypeUsage).get(0)
139             switch (gottenType.type) {
140                 case Type.BOOLEAN: {
141                     hasTryCatch = true
142                     fileContents +=
143         '''
144
145                         try{
146                             boolean __«question.name» =
    Boolean.parseBoolean(br.readLine());
147         '''
148                 }
149                 case Type.NUMBER: {
150                     hasTryCatch = true
151                     fileContents +=
152         '''
153
154                         try{
155                             int __«question.name» =
    Integer.parseInt(br.readLine());
156         '''
157                 }
158                 case Type.TEXT: {
159                     fileContents +=
160         '''
161
162                         String __«question.name» = br.readLine();
163         '''
164                 }
165             }
166         }else if(question.inVar === null && !( question.asValue instanceof
    TypeUsage ) && (question.asValue instanceof FunctionUsage)){
167             val gottenType = ((question.asValue as
    FunctionUsage).exps.get(0) as TypeUsage)
168             val funcName = (question.asValue as FunctionUsage).name.name
169             switch (gottenType.type) {
170                 case Type.BOOLEAN: {
171                     hasTryCatch = true
172                     fileContents +=
173         '''
174
175                         try{
176                             boolean __«question.name» =
    Boolean.parseBoolean(br.readLine());
177         '''
178                 }
179                 case Type.NUMBER: {
```

```
180                     hasTryCatch = true
181                     fileContents +=
182         '''
183
184                         try{
185                             int __«question.name» =
    Integer.parseInt(br.readLine());
186         '''
187                 }
188                 case Type.TEXT: {
189                     fileContents +=
190         '''
191
192                         String __«question.name» = br.readLine();
193         '''
194                 }
195             }
196         }else if(question.inVar !== null && !(question.asValue instanceof
    FunctionUsage)){
197             println("Invar not null")
198             switch ((question.asValue as TypeUsage).type) {
199                 case Type.BOOLEAN: {
200                     hasTryCatch = true
201                     fileContents +=
202         '''
203
204                         try{
205                             «question.inVar.computeExpr» = br.readLine();
206         '''
207                 }
208                 case Type.NUMBER: {
209                     hasTryCatch = true
210                     fileContents +=
211         '''
212
213                         try{
214                             «question.inVar.computeExpr» =
    Integer.parseInt(br.readLine();
215         '''
216                 }
217                 case Type.TEXT: {
218                     fileContents +=
219         '''
220
221                         «question.inVar.computeExpr» = br.readLine();
222         '''
223                 }
224             }
```

```
225         }else if(question.inVar !== null && (question.asValue instanceof
    FunctionUsage)){
226             val gottenType = ((question.asValue as
    FunctionUsage).exps.get(0) as TypeUsage)
227             val funcName = (question.asValue as FunctionUsage).name.name
228             switch (gottenType.type) {
229                 case Type.BOOLEAN: {
230                     hasTryCatch = true
231                     fileContents +=
232         '''
233
234                             try{
235                                 «question.inVar.computeExpr» =
    Boolean.parseBoolean(br.readLine());
236         '''
237                 }
238                 case Type.NUMBER: {
239                     hasTryCatch = true
240                     fileContents +=
241         '''
242
243                             try{
244                                 «question.inVar.computeExpr» =
    Integer.parseInt(br.readLine());
245         '''
246                 }
247                 case Type.TEXT: {
248                     fileContents +=
249         '''
250
251                         «question.inVar.computeExpr» = br.readLine();
252         '''
253                 }
254             }
255         }
256         if(!(question.asValue instanceof TypeUsage) && question.inVar ===
    null){
257             val target = question.targets.get(0)
258             fileContents +=
259         '''
260
261                             if(«computeExpr(question.asValue)»){
262
263                                 nextInteraction="«target.target.name»";
264                                 break ;
265                             }
266         '''
267         }else if(!(question.asValue instanceof TypeUsage) && question.inVar !
```

```
   == null){
268             val target = question.targets.get(0)
269             val funcName = (question.asValue as FunctionUsage).name.name
270             fileContents +=
271         '''
272
273
   if(external.«funcName»(«question.inVar.computeExpr»)){
274
275                                     nextInteraction="«target.target.name»";
276                                     break ;
277                                 }
278         '''
279         }
280         for (target : question.targets) {
281             if (target.logic === null) {
282                 fileContents +=
283         '''
284
285
   nextInteraction="«question.targets.get(0).target.name»";
286                                 break;
287         '''
288             } else {
289                 fileContents +=
290         '''
291
292                                 if(«computeExpr(target.logic)»){
293                                     nextInteraction="«target.target.name»";
294                                     break ;
295                                 }
296         '''
297
298             }
299         }
300
301
302         if (hasTryCatch) {
303             fileContents +=
304         '''
305
306                                 }catch(Exception ex){
307                                     break;
308                                 }
309         '''
310         }
311         return fileContents
312     }
```

```
313
314     def static dispatch String computeExpr(Equals exp) {
315         return '''(«exp.left.computeExpr» == «exp.right.computeExpr»)'''
316     }
317
318     def static dispatch String computeExpr(NotEquals exp) {
319         return '''(«exp.left.computeExpr» != «exp.right.computeExpr»)'''
320     }
321
322     def static dispatch String computeExpr(Less exp) {
323         return '''(«exp.left.computeExpr» < «exp.right.computeExpr»)'''
324     }
325
326     def static dispatch String computeExpr(Greater exp) {
327         return '''(«exp.left.computeExpr» > «exp.right.computeExpr»)'''
328     }
329
330     def static dispatch String computeExpr(LessOrEquals exp) {
331         return '''(«exp.left.computeExpr» <= «exp.right.computeExpr»)'''
332     }
333
334     def static dispatch String computeExpr(GreaterOrEquals exp) {
335         return '''(«exp.left.computeExpr» >= «exp.right.computeExpr»)'''
336     }
337
338     def static dispatch String computeExpr(And exp) {
339         return '''(«exp.left.computeExpr» && «exp.right.computeExpr»)'''
340     }
341
342     def static dispatch String computeExpr(Or exp) {
343         return '''(«exp.left.computeExpr» || «exp.right.computeExpr»)'''
344     }
345
346     def static dispatch String computeExpr(LogicNot exp) {
347         return '''(!«exp.ref.computeExpr»)'''
348     }
349
350     def static dispatch String computeExpr(Parentheses exp) {
351         return '''(«exp.ref.computeExpr»)'''
352     }
353
354     def static dispatch String computeExpr(dk.sdu.frhou18.mdsd.iF22.Boolean
    exp) {
355         return '''(«exp.^val.literal»)'''
356     }
357
358     def static dispatch String computeExpr(Plus exp) {
359         return '''(«exp.left.computeExpr» + «exp.right.computeExpr»)'''
```

```
360     }
361
362     def static dispatch String computeExpr(Minus exp) {
363         return '''(«exp.left.computeExpr» - «exp.right.computeExpr»)'''
364     }
365
366     def static dispatch String computeExpr(Multiplication exp) {
367         return '''(«exp.left.computeExpr» * «exp.right.computeExpr»)'''
368     }
369
370     def static dispatch String computeExpr(Division exp) {
371         return '''(«exp.left.computeExpr» / «exp.right.computeExpr»)'''
372     }
373
374     def static dispatch String computeExpr(MathNumberExp exp) {
375         return '''(«exp.value»)'''
376     }
377
378     def static dispatch String computeExpr(TextExp exp) {
379         return '''«exp.left.computeExpr»+«exp.right.computeExpr»'''
380     }
381
382     def static dispatch String computeExpr(TextLiteral exp) {
383         return '''"«exp.text»"'''
384     }
385
386     def static dispatch String computeExpr(This exp) {
387         return '''(__«EcoreUtil2.getContainerOfType(exp, Question).name»)'''
388     }
389
390     def static dispatch String computeExpr(VarUse exp) {
391         return '''«exp.ref.name»'''
392     }
393
394     def static dispatch String computeExpr(TypeUsage exp){
395         return '''__«EcoreUtil2.getContainerOfType(exp, Question).name»'''
396     }
397
398     def static dispatch String computeExpr(FunctionUsage exp){
399         val params = exp.exps.map[param | param.computeExpr].join(',')
400         return '''external.«exp.name.name»(«params»)'''
401     }
402
403 //  def static dispatch String computeFunction(Function function, EList<Type>
    {
404 //
405 //  }
406
```

```
407     def public static getCorrespondingJavaType(Type type){
408         switch(type){
409             case BOOLEAN: "boolean"
410             case NUMBER: "int"
411             case TEXT: "String"
412         }
413     }
414 }
415
```

```
 1 package dk.sdu.frhou18.mdsd.generator.subgenerators
 2
 3 import org.eclipse.xtext.generator.IFileSystemAccess2
 4 import dk.sdu.frhou18.mdsd.iF22.Model
 5 import org.eclipse.emf.common.util.EList
 6 import dk.sdu.frhou18.mdsd.iF22.Scenario
 7 import dk.sdu.frhou18.mdsd.generator.IF22Generator
 8 import dk.sdu.frhou18.mdsd.iF22.Function
 9 import dk.sdu.frhou18.mdsd.iF22.TypeUsage
10
11 class ExternalGenerator {
12     def static void doGenerate(IFileSystemAccess2 fsa, String pkg, Model
   model, String storyName, EList<Scenario> scenarios){
13         fsa.generateFile('''«pkg»/«IF22Generator.snakeCase(storyName)»/
   External.java''',
14             '''
15                 package «pkg».«IF22Generator.snakeCase(storyName)»;
16
17                 public interface External {
18                 «FOR func : model.functions»
19                     public
   «ScenarioGenerator.getCorrespondingJavaType(func.type)»
   «func.name»(«paramsString(func)»);
20                 «ENDFOR»
21                 }
22             '''
23         )
24     }
25     def static String paramsString(Function func){
26         var StringBuilder sb = new StringBuilder()
27         var counter= 0;
28         for(param : func.parameters.map[p|p as TypeUsage]){
29
   sb.append(ScenarioGenerator.getCorrespondingJavaType(param.type)).append("
   ").append("param").append(counter++)
30         }
31
32         return sb.toString()
33     }
34 }
```