Paula Suárez Prieto, UO269745
Grupo de prácticas PL-02
Curso 2023-2024

Attribute Grammar - Identificación

Attributes

Symbol	Attribute Name	Java Type	Inherited/Synthesized	Description
VarDefinition	scope	ENUM { GLOBAL, LOCAL, PARAMETER }	Inherited	Registro del ámbito en el que se ha definido la variable: - Global: La variable se ha definido en el bloque `global` → `vars` - Local: La variable se ha definido en el bloque `local` dentro de un `feature` - Parameter: La variable es un parámetro de una función
Variable	definition	VarDefinition	Synthesized	Enlace a la definición de esta variable
FunctionDefinition	isBuilder	boolean	Synthesized	True si se ha definido la función previamente en el bloque `create` (constructor). Se usará para verificar que la función puede ser llamada en la llamada `run`.
FunctionCallSent	definition	FunctionDefinition	Synthesized	Enlace a la definición de la función
FunctionCallExpr	definition	FunctionDefinition	Synthesized	Enlace a la definición de la función
FieldDefinition	fieldOwner	StructType	Inherited	Enlace a la struct en la que se define el campo
StructType	definition	StructDefinition	Synthesized	Enlace a la definición de la Struct
RunCall	definition	FunctionDefinition	Synthesized	Enlace a la definición de la función

Paula Suárez Prieto, UO269745
Grupo de prácticas PL-02
Curso 2023-2024

Rules

Node	Predicates	Semantic Functions
program → name:string types:structDefinition* vars:varDefinition* builders:functionBuilder* features:functionDefinition* runCall:runCall	builders == ∅	vars.forEach(v -> v.scope= Scope.GLOBAL)
runCall → name:string args:expression*	functions[name] ≠ ∅ functions[name].isBuilder == TRUE	runCall.definition = functions[name]
structDefinition → name:structType fields:fieldDefinition*	structs[name] == ∅	structs.put(name, structDefinition) fields.forEach(f -> f.fieldOwner = structDefinition) structFields = ∅

functionDefinition → name:string params:varDefinition* returnType:type? vars:varDefinition* sentences:sentence*	functions[name] == ∅	variables.set()
		functions.put(name, functionDefinition)
		
		builders[name] ≠ ∅ then {
		  functionDefinition.isBuilder = true
		  builders.remove(name)
		} else {
		  functionDefinition.isBuilder = false
		}

		params.forEach(p ->
		  p.scope = Scope.PARAMETER
		)
		vars.forEach(v ->
		  v.scope= Scope.LOCAL
		)
		variables.reset()

fieldDefinition → name:string tipo:type	structFields[name] == ∅	structFields.put(name, fieldDefinition)

varDefinition → name:string tipo:type	varDefinition.scope == Scope.GLOBAL then {
	  variables.getFromAny(name)== ∅
	}
	else {
	  variables.getFromTop(name) == ∅
	}
		variables.put(name, varDefinition)

functionBuilder → name:string	builders[name] == ∅	builders[name] = name
functionCallSent:sentence → name:string args:expression*	functions[name] ≠ ∅	functionCallSent.definition = functions[name]
assignment:sentence → left:expression right:expression		
loop:sentence → from:assignment* until:expression body:sentence*		

ifElse:sentence → condition:expression trueBlock:sentence* falseBlock:sentence*		
read:sentence → input:expression*		
print:sentence → op:string input:expression*		
return:sentence → value:expression?		
intConstant:expression → value:string		
realConstant:expression → value:string		
charConstant:expression → value:string		
variable:expression → name:string	variables.getFromAny(name) ≠ ∅	variable.definition = variables.getFromAny(name)
castExpr:expression → castType:type value:expression		
arithmeticExpr:expression → op1:expression operator:string op2:expression		
logicalExpr:expression → op1:expression operator:string op2:expression		
comparationExpr:expression → op1:expression operator:string op2:expression		
minusExpr:expression → op:expression		
notExpr:expression → op:expression		

functionCallExpr:expression → name:string args:expression*	functions[name] ≠ ∅	functionCallExpr.definition = functions[name]
fieldAccess:expression → root:expression field:string		
arrayAccess:expression → array:expression index:expression		
intType:type → ε		
doubleType:type → ε		
charType:type → ε		
voidType:type → ε		
structType:type → name:string	structs[name] ≠ ∅	structType.definition = structs[name]
arrayType:type → dimension:intConstant tipo:type		

Operators samples (cut & paste if needed):
⇒ ⇔ ≠ ∅ ∈ ∉ ∪ ⋂ ⊂ ⊄ ∑ ∃ ∀

Auxiliary Data Structures

Symbol	Java Type	Description
variables	ContextMap<String, VarDefinition>	Conjunto de las variables declaradas

builders	List<String>	Lista de los nombres declarados en el bloque ‘create’, es decir, los constructores declarados (FunctionBuilder)
functions	Map<String, FunctionDefinition>	Conjunto de las funciones definidas
structs	Map<String, StructDefinition>	Conjunto de los structs (defTuple) declarados
structFields	Map<String, FieldDefinition>	Conjunto de los campos declarados en un struct. Está lista se vacía cada vez que se termina de visitar una definición de Struct.