## Attribute Grammar

### Attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Symbol | Attribute Name | Java Type | Inherited/Synthesized | Description |
| Expression | Type | Type | Synthesized |  |
| Expression | Lvalue | Boolean | Synthesized |  |
|  |  |  |  |  |

### Rules

|  |  |  |
| --- | --- | --- |
| Node | Predicates | Semantic Functions |
| program → classDeclaration runStatement |  |  |
| classDeclaration → name:string globalSection? createSection featureSection\* |  |  |
| globalSection → typesSection? varSection? |  |  |
| typesSection → structDeclaration\* |  |  |
| varSection → variableDeclaration\* |  |  |
| variableDeclaration:declaration → identifiers:string\* type |  |  |
| structDeclaration → name:string structField\* |  |  |
| structField:declaration → name:string type |  |  |
| createSection → string\* |  |  |
| featureSection → name:string args? type? localSection? statement\* | Parámetros deben ser tipo simple  Tipo void -> no return  No tipo void -> return | Sentencias.decFeature=feature |
| localSection → variableDeclaration\* |  |  |
| args → arg\* |  |  |
| arg:declaration → name:string type |  |  |
| assignment:statement → left:expression right:expression | ambos lados sean de tipos simples. Ambos tipos iguales.  Left.lvalue=true |  |
| print:statement → expression\* | expresión\* tipo simple |  |
| println:statement → expression\* | expresión\* tipo simple |  |
| read:statement → expression\* | expresión\* tipo simple  lvalue=true |  |
| bloqueif:statement → expression st2:statement\* st3:statement\* | Condición tipo int |  |
| loopFrom:statement → st1:statement\* expression body:statement\* | expresión\* tipo simple |  |
| return:statement → expression? | Return.type= función.type |  |
| functionCallStatement:statement → name:string expression\* | numParam=función.numArgs  Tipos params correctos | functionCallStatement.type=functionCallStatement.definition.type |
| runStatement:statement → name:string expression\* | numParam=función.numArgs  Tipos params correctos |  |
| intLiteral:expression → name:string |  | Type=intType  Lvalue=false |
| realConstant:expression → name:string |  | Type=DoubleType  Lvalue=false |
| charConstant:expression → name:string |  | Type=CharacterType  Lvalue=false |
| functionCallExp:expression → name:string expresiones:expression\* | numParam=función.numArgs  Tipos params correctos | functionCallExp.type=functionCallExp.definition.type |
| arrayAcces:expression → exp2:expression exp3:expression | Exp2.lvalue=true  Exp3.tipo=intType |  |
| variableAcces:expression → name:string |  | variableAcces.type=variableAcces.definition.type  variableAcces.lvalue=true |
| restaUnaria:expression → exp2:expression | Exp2.type= intType | doubleType | restaUnaria.type= exp2.type |
| parentesis:expression → exp2:expression |  | Paréntesis.type=exp2.type  Paréntesis.lvalue=exp2.lvalue |
| negacion:expression → exp2:expression | Exp2.type= intType | Negación.lvalue=exp2.lvalue  Negación.type=exp2.type |
| cast:expression → tipoCast:type exp2:expression | INTEGER -> (INTEGER, DOUBLE, CHARACTER)  DOUBLE -> (INTEGER, DOUBLE)  CHARACTER -> (INTEGER, CHARACTER) | Cast.type= toType.type  Lvalue=false |
| arithmetic:expression → exp2:expression name:string exp3:expression | sameType(left.type,right.type)==true | expression.type= left.type  arithmetic.lvalue=false |
| relacional:expression → exp2:expression name:string exp3:expression | Left.type== intType | left.type=doubleType left.type=right.type | Relacional.type=left.type  Lvalue=false |
| booleanExp:expression → exp2:expression name:string exp3:expression | Left.type&& right.type== intType | Lvalue=false |
| structFieldAcces:expression → exp2:expression name:string |  | structFieldAcces.type=exp2.dot(structfieldAcces.name) |
| intType:type → name:string |  |  |
| doubleType:type → name:string |  |  |
| characterType:type → name:string |  |  |
| identType:type → name:string |  |  |
| arraytype:type → intValue:int type2:type |  |  |
| errorType:type → name:string |  |  |
| voidType:type → name:string |  |  |

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