# Code Specification

|  |  |  |
| --- | --- | --- |
| **Nodo** | **Predicados** | **Reglas Semánticas** |
| **program** → *classnode*:classNode *methodcallsentence*:methodCallSentence |  |  |
| **classNode** → *name*:String *definition*:definition\* *createMethod*:String\* *method*:method\* | tabSimbFeature[createMethod] ≠ ∅ | ClassNode.createDefinitions = tabSimbFeature[createMethod] |
| **method** → *name*:String *parameter*:parameter\* *retorno*:type *definition*:definition\* *sentence*:sentence\* | tabSimbFeature[name] == ∅  si retorno ≠ voidType  returnNode ⊂ method.definition | tabSimbFeature[name] = method |
| **parameter** → *name*:String *type*:type | tabSimbVar[name] == ∅  esPrimitivo(type) | tabSimbVar[name] = parameter |
|  |  |  |
| **print**:sentence → *string*:String *expr*:expr\* | esPrimitivo(expri.type) |  |
| **read**:sentence → *expr*:expr\* | esPrimitivo(expri.type)  expr.modifiable |  |
| **assignment**:sentence → *left*:expr *right*:expr | mismoTipo(left.type, right.type)  left.modifiable |  |
| **conditional**:sentence → *condition*:expr *iftrue*:sentence\* *iffalse*:sentence\* | condition.type == intType |  |
| **loop**:sentence → *init*:sentence\* *condition*:expr *sentence*:sentence\* | condition.type == intType |  |
| **returnNode**:sentence → *expr*:expr | si returnNode.method.retorno == voidType  expr == null  si no  expr.type == returnNode.method.retorno | returnNode.method = method |
| **methodCallSentence**:sentence → *name*:String *args*:expr\* | tabSimbFeature[name] ≠ ∅  methodCallSentence.definition.parameter.length == args.length | MethodCallSentence.definition = tabSimbFeature[name] |
|  |  |  |
| **exprBinariaAritmetica**:expr → *left*:expr *op*:String *right*:expr | mismoTipo(left.type, right.type)  left.type == intType || left.type == realType | exprBianariaAritmetica.type = left.type  exprBinariaAritmetica.modifiable = false |
| **exprUnariaAritmetica**:expr → *op*:String *expr*:expr | expr.type == intType || expr.type == realType | exprUnariaAritmetica.type = expr.type  exprUnariaAritmetica.modifiable = false |
| **exprBinariaLogica**:expr → *left*:expr *op*:String *right*:expr | mismoTipo(left.type, right.type)  si(“<>” || “=”)  esPrimitivo(left.type)  sino  esNumero(left.type) | exprBinariaLogica.type = left.type  exprBinariaLogica.modifiable = false |
| **exprUnariaLogica**:expr → *op*:String *expr*:expr | expr.type == intType | exprUnariaLogica.type = expr.type  exprUnariaLogica.modifiable = false |
| **access**:expr → *left*:expr *op*:String *right*:String | left.type == structType | acces.type = left.type |
| **arrayAcces**:expr → *left*:expr *right*:expr | left.type == arrayType  right.type == intType | arrayAcces.typeOfElements =  arrayAcces.type = left.type.type  arrayAcces.modifiable = true |
| **cast**:expr → *typeToConvert*:type *expr*:expr | esPrimitivo(typeToConvert)  typeToConvert ≠ expr.type  esPrimitivo(expr)  isCompatibleCast(typeToConvert, expr) | cast.type = typeToConvert  cast.modifiable = false |
| **litEnt**:expr → *string*:String |  | litEnt.type = intType  litEnt.modifiable = false |
| **litReal**:expr → *string*:String |  | litReal.type = realType  litReal.modifiable = false |
| **litChar**:expr → *string*:String |  | litChar.type = charType  litChar.modifiable = false |
| **variable**:expr → *string*:String | tabSimbVar[name] ≠ ∅ | Variable.definition = tabSimbFeature[name]  variable.type = variable.definition.type  variable.modifiable = true |
| **methodCallExpr**:expr → *name*:String *args*:expr\* | tabSimbFeature[name] ≠ ∅  methodCallExpr.definition.parameter.length == args.length  methodCallExpr.definition.type != voidType | MethodCallExpr.definition = tabSimbFeature[name]  methodCallExpr.type = methodCallExpr.definition.type |
|  |  |  |
| **intType**:type → λ |  |  |
| **realType**:type → λ |  |  |
| **charType**:type → λ |  |  |
| **arrayType**:type → *index*:int *type*:type |  |  |
| **voidType**:type → λ |  |  |
| **structType**:type → *name*:String | tabSimbTuple[name] ≠ ∅ | StructType.definition = tabSimbTuple[name] |
|  |  |  |
| **tupleDefinition**:definition → *name*:String *vardefinition*:varDefinition\* | tabSimbTuple[name] == ∅ | tabSimbTuple[name] = tupleDefinition |
| **varDefinition**:definition → *name*:String\* *type*:type | tabSimbVar[name] == ∅ | tabSimbVar[name] = varDefinition |
|  |  |  |

Recordatorio de los operadores (para cortar y pegar): ⇒ ⇔ ≠ ∅ ∈ ∉ ∪ ∩ ⊂ ⊄ ∑ ∃ ∀

# Atributos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nodo/Categoría Sintáctica** | **Nombre del Atributo** | **Tipo Java** | **Heredado/Sintetizado** | **Descripción** |
| Variable | definition | VarDefinition | Sintetizado | Nodo donde está definida la variable |
| StructType | definition | TupleDefinition | Sintetizado | Nodo donde está definida la estructura |
| MethodCallSentence | definition | Method | Sintetizado | Nodo donde está definido el método |
| MethodCallExpr | definition | Method | Sintetizado | Nodo donde está definido el método |
| Parameter | definition | VarDefinition | Sintetizado | Nodo donde está definido el parámetro |
| ClassNode | createDefinitions | List<Method> | Sintetizado | Nodo donde está definido el método constructor |
| Expr | type | Type | Sintetizado | Para que la expresión tenga su propio tipo |
| Expr | modifiable | Boolean | Sintetizado | Para saber si la expresión es modificable |
| ReturnNode | metodo | Method | Heredado | Asocia la sentencia de retorno con su método |
| ArrayAcces | typeOfElements | Type | Sintetizado | Para saber el tipo de los elementos del array |

# Estructura de Datos Auxiliares

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Tipo Java** | **Descripción** |
| tabSimbTuple | HashMap<String, TupleDefinition> | Tabla hash que almacenará las definiciones de las tuplas a medida que se vayan encontrando. |
| tabSimbVar | ContextMap<String, VarDefinition> | Tabla hash con contextos que almacenará las definiciones de las variables normales a medida que se vayan encontrando. |
| tabSimbFeature | HashMap<String, Method> | Tabla hash que almacenará las definiciones de los métodos a medida que se vayan encontrando. |

# Funciones Auxiliares

boolean mismoTipo(typeA, typeB) {typeA == typeB}

boolean esPrimitivo(tipo) {tipo == int || tipo == real || tipo == char}

boolean esNumero(tipo){tipo == int || tipo == real}

boolean isCompaitbleCast(typeToConvert, type) {

if(typeToConvert == type)

return false;

else if(typeToConvert == real && type == char)

return false;

else if(typeToConvert == char && type == real)

return false;

else

return true;

}