MANUAL TÉCNICO

ANALIZADOR DE LENGUAJE PASCAL: PROYECTO DE ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2

BREVE DESCRIPCIÓN

El presente proyecto, es una implementación de las 3 primeras fases de compilación de un lenguaje de programación, en este caso, Pascal, y también provee funcionalidades extra como revisar tablas de tipos y tablas de símbolos.

HERRAMIENTAS UTILIZADAS

- ## LENGUAJE DE PROGRAMACIÓN JAVA - VERSIÓN 21

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems.

El lenguaje Java proporciona:

- El paradigma de la programación orientada a objetos.
- Ejecución de un mismo programa en múltiples sistemas operativos y plataformas.
- Es posible utilizarlo para múltiples propósitos, desde aplicaciones de escritorio hasta en servidores web.
- Tiene una curva de aprendizaje media pero también toma lo mejor de otros lenguajes orientados a objetos, como C++.

También se utilizaron librerías/bibliotecas externas como:

- **JFlex - Análisis léxico**
- **Java CUP - Análisis sintáctico**

● JAVA SWING: INTERFAZ DE USUARIO

Swing es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, listas desplegables y tablas.
Desde sus inicios el entorno Java ya contaba con una biblioteca de componentes gráficos conocida como AWT. Esta biblioteca estaba concebida como una API estandarizada que permitía utilizar los componentes nativos de cada sistema operativo. Entonces una aplicación Java corriendo en Microsoft Windows usaría el botón estándar de Windows y una aplicación corriendo en UNIX usaría el botón estándar de Motif. En la práctica esta tecnología no funcionó:

Al depender fuertemente de los componentes nativos del sistema operativo, el programador de AWT estaba confinado al máximo denominador común entre ellos. Es decir que sólo se dispone en AWT de las funcionalidades comunes en todos los sistemas operativos.
El comportamiento de los controles varía mucho de sistema a sistema y se vuelve muy difícil construir aplicaciones portables. Fue por esto que el eslogan de Java "Escríbalo una vez, ejecútelo en todos lados" fue parodiado como "Escríbalo una vez, pruébelo en todos lados".

● IDE: INTELLIJ IDEA ULTIMATE

IntelliJ IDEA es un entorno de desarrollo integrado (IDE) escrito en Java para desarrollar software informático escrito en Java, Kotlin, Groovy y otros lenguajes basados en la JVM. Está desarrollado por JetBrains (anteriormente conocido como IntelliJ) y está disponible como una edición comunitaria con licencia de Apache 2, y en una edición comercial patentada. Ambos se pueden utilizar para el desarrollo comercial.

● GITHUB

Es un sistema de control de versiones de código y gestión de proyectos, a su vez también funciona como una plataforma de estilo red social diseñada para desarrolladores para poder compartir código entre más personas y colaborar en el mismo. Se utilizó GitFlow como metodología

- ## MACOS v13 - SISTEMA OPERATIVO

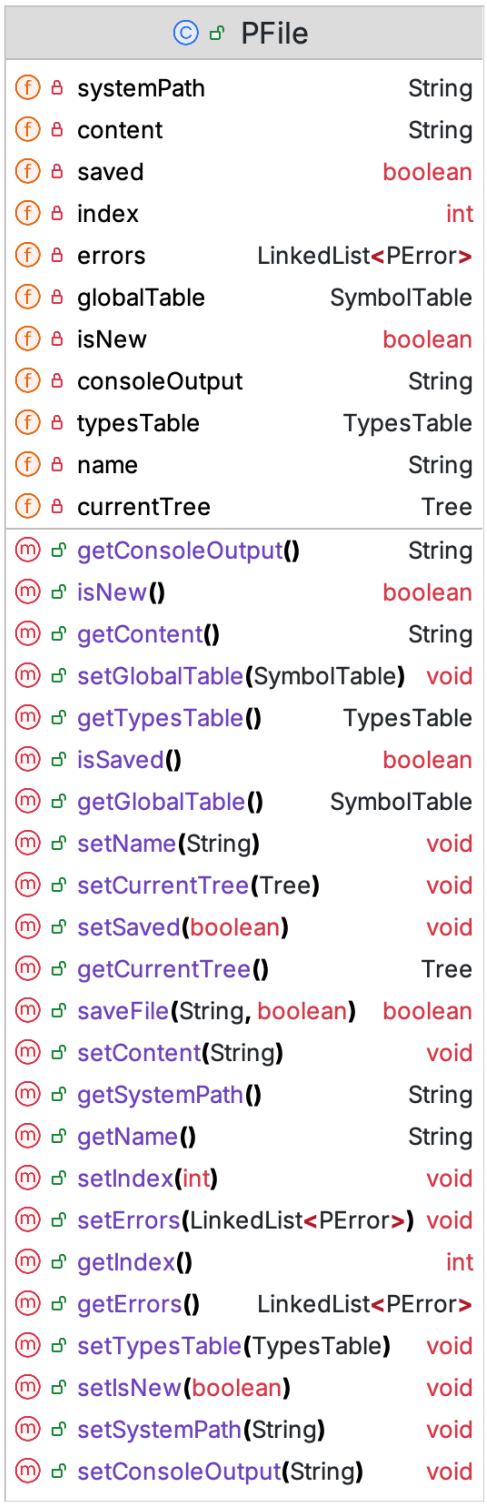
macOS (previamente Mac OS X, luego OS X) es una serie de sistemas operativos gráficos desarrollados y comercializados por Apple desde 2001. Es el sistema operativo principal para la familia de computadoras Mac de Apple. Dentro del mercado de computadoras de escritorio, portátiles, hogareñas y mediante el uso de la web.

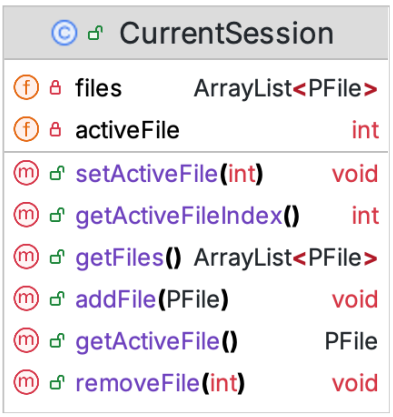
macOS se basa en tecnologías desarrolladas entre 1985 y 1997 en NeXT. Se logró la certificación UNIX 03 para la versión Intel de Mac OS X 10.5 Leopard y todos los lanzamientos de Mac OS X 10.6 Snow Leopard hasta la versión actual también tienen la certificación UNIX 03. macOS comparte su núcleo basado en Unix, llamado Darwin, y muchos de sus frameworks con iOS 16, tvOS y watchOS.

DIAGRAMA DE CLASES

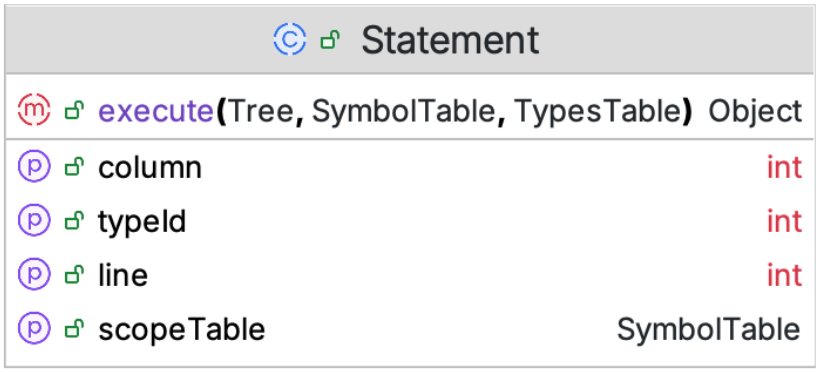
Analysis

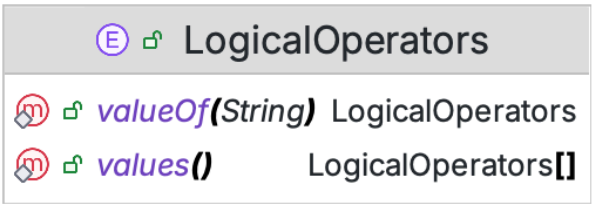
Data

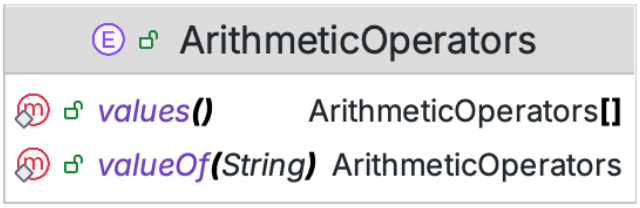

PFile
systemPath String
content String
saved boolean
index int
errors LinkedList<PError>
globalTable SymbolTable
isNew boolean
consoleOutput String
typesTable TypesTable
name String
currentTree Tree
getConsoleOutput() String
isNew() boolean
getContent() String
setGlobalTable(SymbolTable) void
getTypesTable() TypesTable
isSaved() boolean
getGlobalTable() SymbolTable
setName(String) void
setCurrentTree(Tree) void
setSaved(boolean) void
getCurrentTree() Tree
saveFile(String, boolean) boolean
setContent(String) void
getSystemPath() String
getName() String
setIndex(int) void
setErrors(LinkedList<PError>) void
getIndex() int
getErrors() LinkedList<PError>
setTypesTable(TypesTable) void
setIsNew(boolean) void
setSystemPath(String) void
setConsoleOutput(String) void

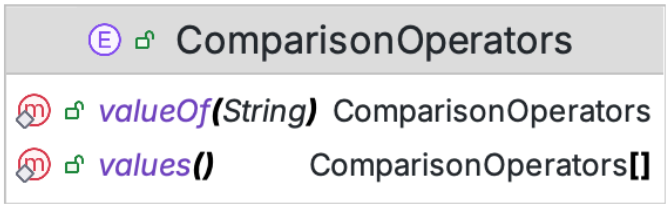

CurrentSession
files ArrayList<PFile>
activeFile int
setActiveFile(int) void
getActiveFileIndex() int
getFiles() ArrayList<PFile>
addFile(PFile) void
getActiveFile() PFile
removeFile(int) void

Base

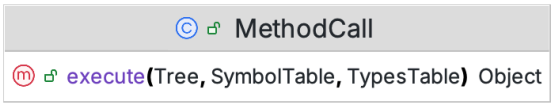

Statement
execute(Tree, SymbolTable, TypesTable) Object
column int
typeId int
line int
scopeTable SymbolTable

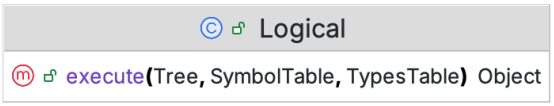

LogicalOperators
valueOf(String) LogicalOperators
values() LogicalOperators[]

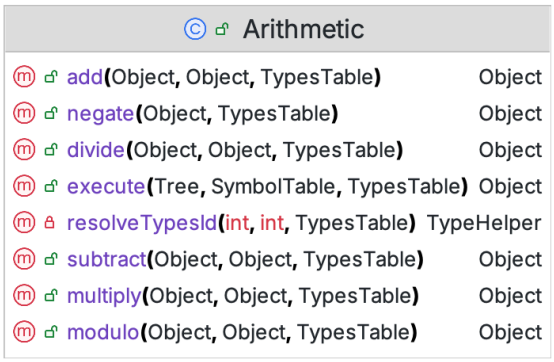

ArithmeticOperators
values() ArithmeticOperators[]
valueOf(String) ArithmeticOperators

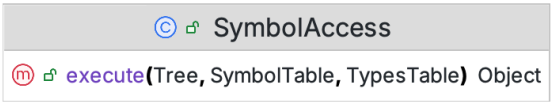

ComparisonOperators
valueOf(String) ComparisonOperators
values() ComparisonOperators[]

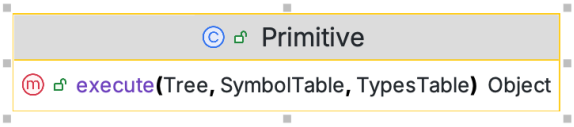
Expressions

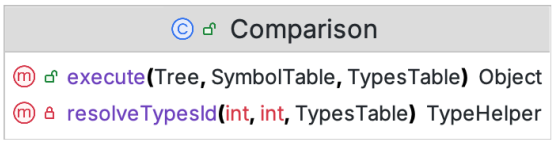


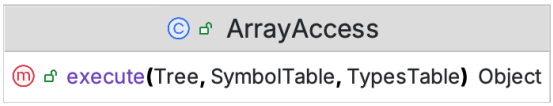




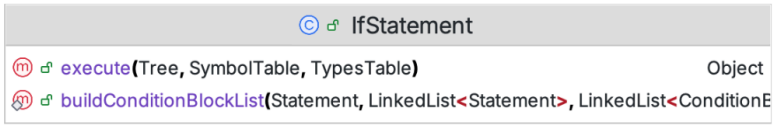


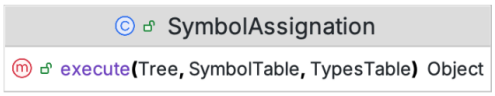


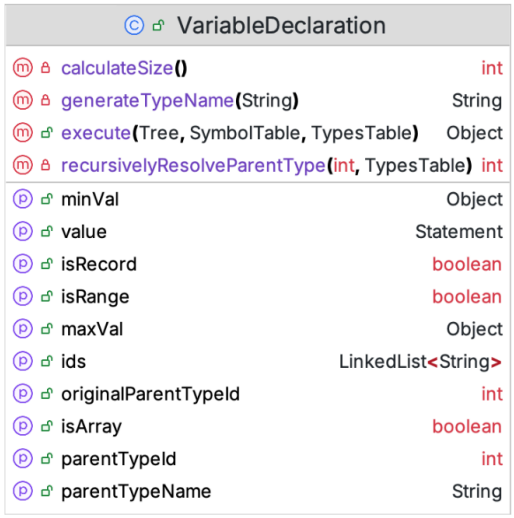


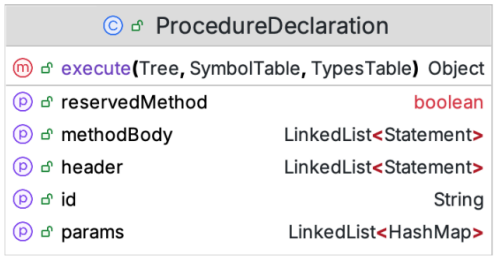


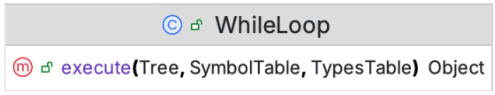
Statements

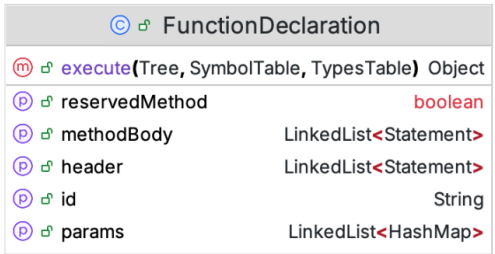

IfStatement
execute(Tree, SymbolTable, TypesTable) Object
buildConditionBlockList(Statement, LinkedList<Statement>, LinkedList<ConditionE

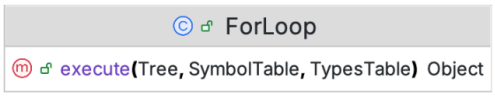

SymbolAssignation
execute(Tree, SymbolTable, TypesTable) Object

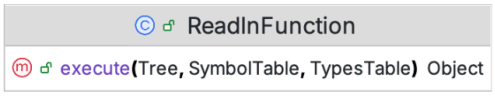

VariableDeclaration
calculateSize() int
generateTypeName(String) String
execute(Tree, SymbolTable, TypesTable) Object
recursivelyResolveParentType(int, TypesTable) int
minVal Object
value Statement
isRecord boolean
isRange boolean
maxVal Object
ids LinkedList<String>
originalParentTypeId int
isArray boolean
parentTypeId int
parentTypeName String

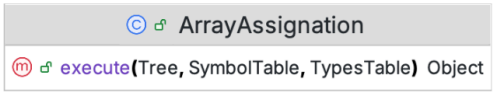

ProcedureDeclaration
execute(Tree, SymbolTable, TypesTable) Object
reservedMethod boolean
methodBody LinkedList<Statement>
header LinkedList<Statement>
id String
params LinkedList<HashMap>


WhileLoop
execute(Tree, SymbolTable, TypesTable) Object

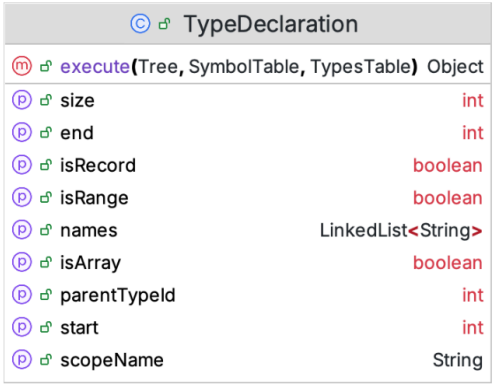

FunctionDeclaration
execute(Tree, SymbolTable, TypesTable) Object
reservedMethod boolean
methodBody LinkedList<Statement>
header LinkedList<Statement>
id String
params LinkedList<HashMap>

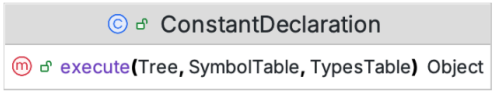

ForLoop
execute(Tree, SymbolTable, TypesTable) Object

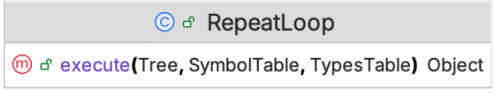

ReadInFunction
execute(Tree, SymbolTable, TypesTable) Object


ArrayAssignation
execute(Tree, SymbolTable, TypesTable) Object

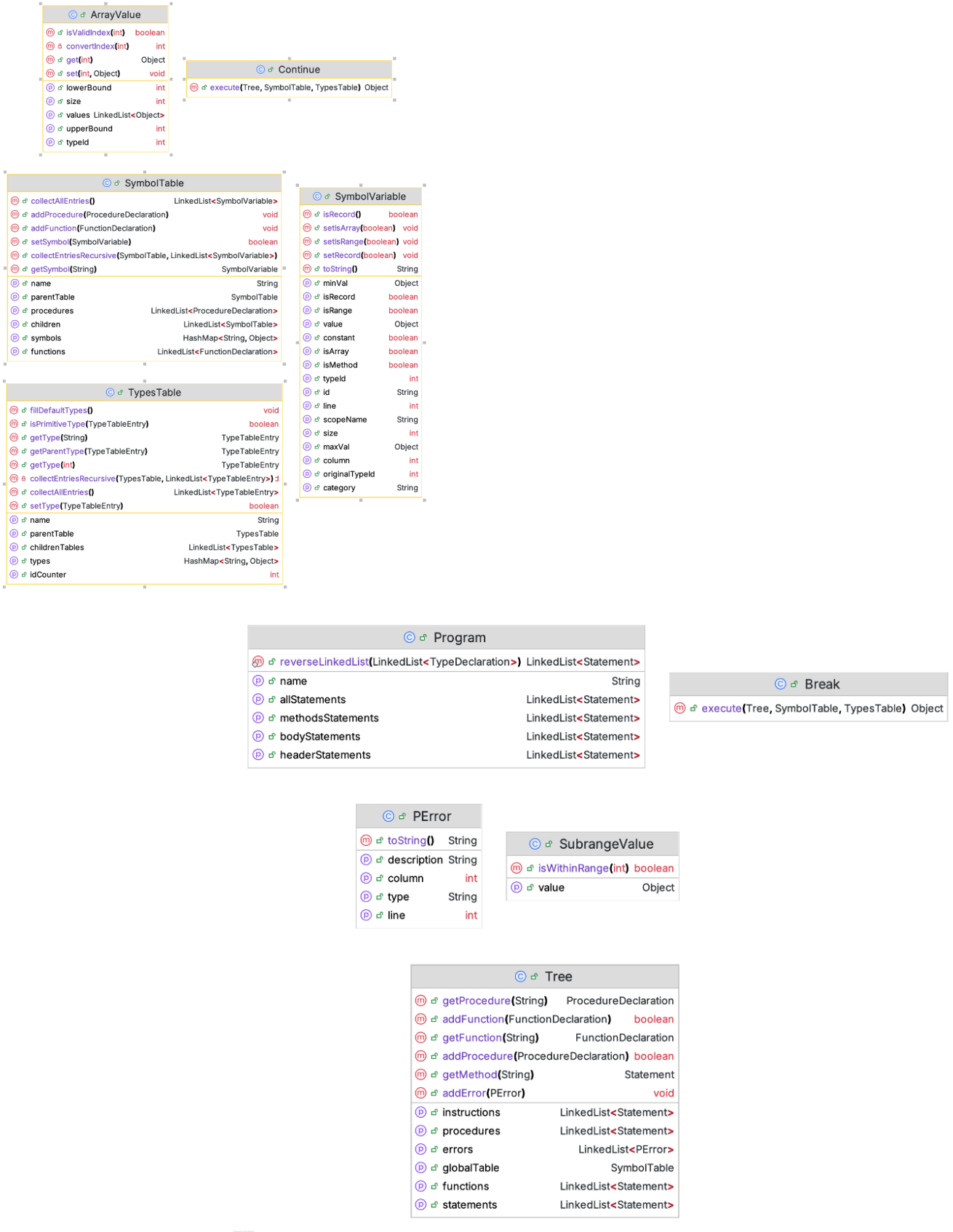

WriteInFunction
execute(Tree, SymbolTable, TypesTable) Object


TypeDeclaration
execute(Tree, SymbolTable, TypesTable) Object
size int
end int
isRecord boolean
isRange boolean
names LinkedList<String>
isArray boolean
parentTypeId int
start int
scopeName String

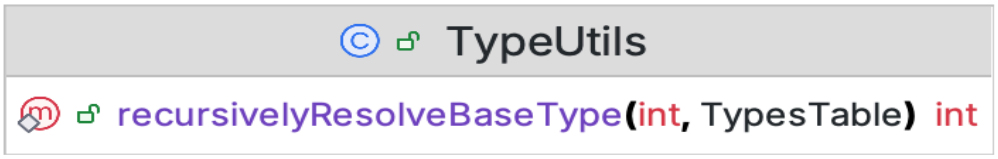

ConstantDeclaration
execute(Tree, SymbolTable, TypesTable) Object

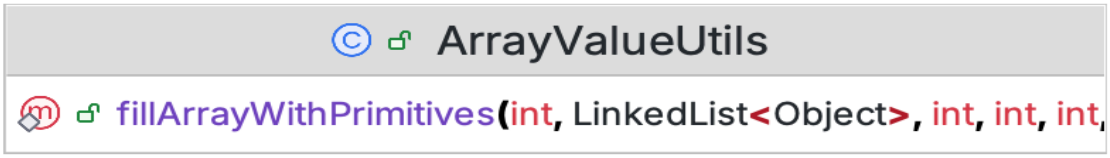

RepeatLoop
execute(Tree, SymbolTable, TypesTable) Object

Structs

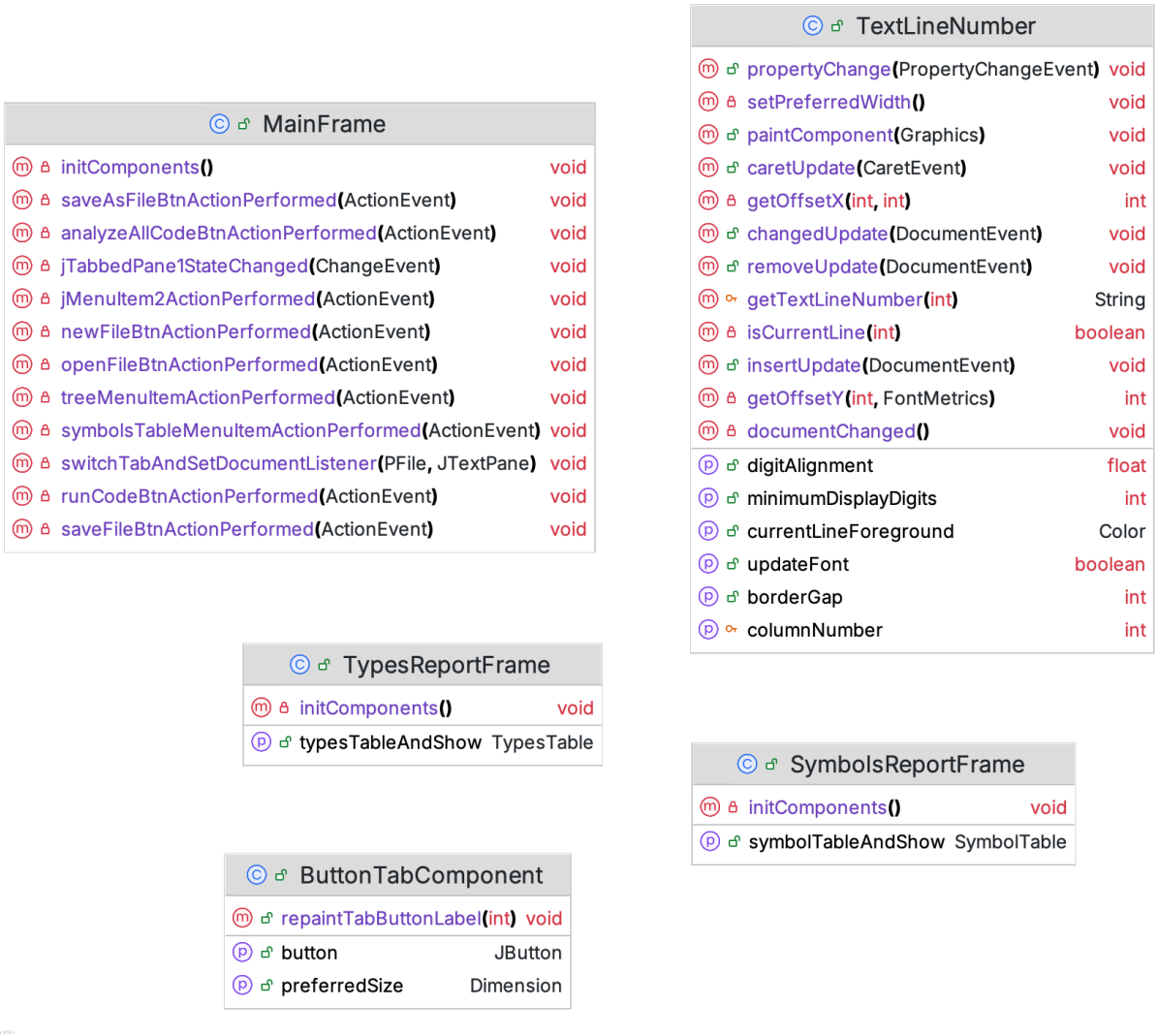

ArrayValue
isValidIndex(int) boolean
convertIndex(int) int
get(int) Object
set(int, Object) void
lowerBound int
size int
values LinkedList<Object>
upperBound int
typeId int
Continue
execute(Tree, SymbolTable, TypesTable) Object
SymbolTable
collectAllEntries() LinkedList<SymbolVariable>
addProcedure(ProcedureDeclaration) void
addFunction(FunctionDeclaration) void
setSymbol(SymbolVariable) boolean
collectEntriesRecursive(SymbolTable, LinkedList<SymbolVariable>)
getSymbol(String) SymbolVariable
name String
parentTable SymbolTable
procedures LinkedList<ProcedureDeclaration>
children LinkedList<SymbolTable>
symbols HashMap<String, Object>
functions LinkedList<FunctionDeclaration>
SymbolVariable
isRecord() boolean
setIsArray(boolean) void
setIsRange(boolean) void
setRecord(boolean) void
toString() String
minVal Object
isRecord boolean
isRange boolean
value Object
constant boolean
isArray boolean
isMethod boolean
typeId int
id String
line int
scopeName String
size int
maxVal Object
column int
originalTypeId int
category String
TypesTable
fillDefaultTypes() void
isPrimitiveType(TypeTableEntry) boolean
getType(String) TypeTableEntry
getParentType(TypeTableEntry) TypeTableEntry
getType(int) TypeTableEntry
collectEntriesRecursive(TypesTable, LinkedList<TypeTableEntry>)
collectAllEntries() LinkedList<TypeTableEntry>
setType(TypeTableEntry) boolean
name String
parentTable TypesTable
childrenTables LinkedList<TypesTable>
types HashMap<String, Object>
idCounter int
Program
reverseLinkedList(LinkedList<TypeDeclaration>) LinkedList<Statement>
name String
allStatements LinkedList<Statement>
methodsStatements LinkedList<Statement>
bodyStatements LinkedList<Statement>
headerStatements LinkedList<Statement>
Break
execute(Tree, SymbolTable, TypesTable) Object
PError
toString() String
description String
column int
type String
line int
SubrangeValue
isWithinRange(int) boolean
value Object
Tree
getProcedure(String) ProcedureDeclaration
addFunction(FunctionDeclaration) boolean
getFunction(String) FunctionDeclaration
addProcedure(ProcedureDeclaration) boolean
getMethod(String) Statement
addError(PError) void
instructions LinkedList<Statement>
procedures LinkedList<Statement>
errors LinkedList<PError>
globalTable SymbolTable
functions LinkedList<Statement>
statements LinkedList<Statement>

Utils


TypeUtils
recursivelyResolveBaseType(int, TypesTable) int


ArrayValueUtils
fillArrayWithPrimitives(int, LinkedList<Object>, int, int, int,

View


MainFrame
initComponents() void
saveAsFileBtnActionPerformed(ActionEvent) void
analyzeAllCodeBtnActionPerformed(ActionEvent) void
jTabbedPane1StateChanged(ChangeEvent) void
jMenuItem2ActionPerformed(ActionEvent) void
newFileBtnActionPerformed(ActionEvent) void
openFileBtnActionPerformed(ActionEvent) void
treeMenuItemActionPerformed(ActionEvent) void
symbolsTableMenuItemActionPerformed(ActionEvent) void
switchTabAndSetDocumentListener(PFile, JTextPane) void
runCodeBtnActionPerformed(ActionEvent) void
saveFileBtnActionPerformed(ActionEvent) void
TextLineNumber
propertyChange(PropertyChangeEvent) void
setPreferredWidth() void
paintComponent(Graphics) void
caretUpdate(CaretEvent) void
getOffsetX(int, int) int
changedUpdate(DocumentEvent) void
removeUpdate(DocumentEvent) void
getTextLineNumber(int) String
isCurrentLine(int) boolean
insertUpdate(DocumentEvent) void
getOffsetY(int, FontMetrics) int
documentChanged() void
digitAlignment float
minimumDisplayDigits int
currentLineForeground Color
updateFont boolean
borderGap int
columnNumber int
TypesReportFrame
initComponents() void
typesTableAndShow TypesTable
SymbolsReportFrame
initComponents() void
symbolTableAndShow SymbolTable
ButtonTabComponent
repaintTabButtonLabel(int) void
button JButton
preferredSize Dimension

CLASES Y ALGORITMOS

Analysis:

Las clases dentro del paquete de ***Analysis***, funciona puramente para la generación del código del analizador léxico y sintáctico proveído por las anteriormente mencionadas librerías, JFlex y CUP. Cuenta con una función *main* que hace uso de las librerías junto con los archivos de configuración.

Data:

Estas clases únicamente son utilizadas para representar los archivos abiertos dentro del el programa y sus propiedades para poder manejar la interfaz de usuario, acciones como guardar, cerrar o abrir nuevos archivos se manejan con estas clases

Base:

Las clases dentro de este paquete solamente sirven como estructura base para las demás estructuras de datos que se utilizan a lo largo del programa. Proporcionan propiedades básicas y algunas constantes predefinidas

Expressions:

Estas clases tienen como fin representar los valores que pueda llegar a contener una expresión dentro del lenguaje de programación que se esta analizando, y tabmién de interpretarlas y validarlas a lo largo que se analiza o analizan los diferentes archivos.

Statements:

Las clases dentro de este paquete al igual que Expressions tienen como fin representar las sentencias que pueden llegar a definirse y a utilizar dentro del análisis de este lenguaje, y no necesariamente retornan un valor, haciendo que estas sean una especie de envoltorio para expresiones y otras validaciones

Struct:
Las clases dentro de este paquete se utilizan para guardar diferentes tipos de datos utilizando diferentes tipos de estructuras de datos personalizadas para poder llevar control de diferentes elementos/miembros dentro del análisis del lenguaje. Elementos como los símbolos en la tabla de símbolos y los tipos de datos.

Utils:
Clases que proporcionan utilidades varias

View:
Clases que proporcionan la funcionalidad de la interfaz gráfica a través de la librería Java Swing

GRÁMATICAS EN FORMATO BNF

```
<start> ::= <program> "."
          | <program> .

<program> ::= <program_header> <procedure_function_declarations> "begin" <program_body> "end"
            | <program_header> "begin" <program_body> "end"
            | <program_header> <procedure_function_declarations> "begin" "end"
            | <program_header> "begin" "end" .

<procedure_function_declarations> ::= <procedure_function_declarations>
<procedure_function_declaration>
                                    | <procedure_function_declaration> .

<procedure_function_declaration> ::= <function_declaration_block>
                                   | <procedure_declaration_block> .

<program_header> ::= "program" <id> ";" <declarations_block> .

<declarations_block> ::= <type_declaration_block> <constant_declaration_block>
<variable_declaration_block> .

<program_body> ::= <program_body> <program_body_statement>
                 | <program_body_statement> .

<program_body_statement> ::= <symbol_assignation> ";"
                           | <array_assignation> ";"
                           | <if_statement>
                           | <while_statement>
                           | <repeat_statement>
                           | <for_statement>
                           | "break" ";"
                           | "continue" ";"
                           | <call_to_method> ";"
                           | <case_statement>
                           | <writeln_call> ";"
                           | <readln_call> ";" .

<statement> ::= <symbol_assignation>
              | <array_assignation>
              | <if_statement>
              | <case_statement>
              | <while_statement>
              | <repeat_statement>
              | <for_statement>
              | "break"
              | "continue"
              | <call_to_method>
              | <writeln_call>
              | <readln_call> .

<type_declaration_block> ::= "type" <type_declaration_list>
```

```
                          | .

<type_declaration_list> ::= <type_declaration>
                          | <type_declaration> ";"
                          | <type_declaration> ";" <type_declaration_list> .

<type_declaration> ::= <identifier_list> "=" <type>
                     | <identifier_list> "=" <array_type>
                     | <identifier_list> "=" <range_def>
                     | <identifier_list> "=" <id>
                     | <record_declaration> .

<identifier_list> ::= <identifier_list> "," <id>
                    | <id> .

<type> ::= "integer"
         | "real"
         | "char"
         | "string"
         | "boolean" .

<range_int> ::= <integer>
              | "-" <integer> .

<range_def> ::= <range_int> ".." <range_int> .

<array_type> ::= "array" "[" <range_def> "]" "of" <type>
               | "packed" "array" "[" <range_def> "]" "of" <type>
               | "array" "[" <range_def> "]" "of" <id>
               | "packed" "array" "[" <range_def> "]" "of" <id> .

<record_declaration> ::= <identifier_list> "=" <record_use> .

<record_use> ::= "record" <field_list> "end" .

<field_list> ::= <field_declaration>
               | <field_declaration> ";"
               | <field_declaration> ";" <field_list> .

<field_declaration> ::= <identifier_list> ":" <type>
                      | <identifier_list> ":" <array_type>
                      | <identifier_list> ":" <range_def>
                      | <identifier_list> ":" "record" <id> .

<constant_declaration_block> ::= "const" <constant_declaration_list>
                               | .

<constant_declaration_list> ::= <constant_declaration> ";" <constant_declaration_list>
                              | <constant_declaration>
                              | .

<constant_declaration> ::= <id> "=" <constant_value> .
```

```
<constant_value> ::= <integer>
                   | <decimal>
                   | <char>
                   | <string>
                   | "true"
                   | "false" .

<variable_declaration_block> ::= "var" <variable_declaration_list>
                                | .

<variable_declaration_list> ::= <variable_declaration> ";" <variable_declaration_list>
                              | <variable_declaration>
                              | .

<variable_declaration> ::= <identifier_list> ":" <type>
                         | <identifier_list> ":" <id>
                         | <identifier_list> ":" <array_type>
                         | <identifier_list> ":" <record_use>
                         | <identifier_list> ":" <range_def> .

<symbol_assignation> ::= <id> ":=" <expression> .

<array_assignation> ::= <id> "[" <expression> "]" ":=" <expression> .

<case_value_list> ::= <case_value_list> "," <expression>
                    | <expression> .

<case_else> ::= "else" <statement> ";"
              | "else" "begin" <program_body> "end" .

<while_statement> ::= "while" "(" <expression> ")" "do" <statement> ";"
                    | "while" "(" <expression> ")" "do" "begin" <program_body> "end" ";" .

<repeat_statement> ::= "repeat" <program_body> "until" <expression> ";" .

<for_statement> ::= "for" <id> ":=" <expression> "to" <expression> "do" <statement> ";"
                  | "for" <id> ":=" <expression> "to" <expression> "do" "begin" <program_body>
"end" ";" .

<function_declaration_start> ::= "function" <id> "(" <parameters_declaration> ")" ":" <type> ";"
                               | "function" <id> "(" <parameters_declaration> ")" ":" <id> ";"
                               | "function" <id> ":" <type> ";" .

<function_declaration_block> ::= <function_declaration_start> <declarations_block> "begin"
<program_body> "end" ";" .

<procedure_declaration_start> ::= "procedure" <id> "(" <parameters_declaration> ")" ";" .

<procedure_declaration_block> ::= <procedure_declaration_start> <declarations_block> "begin"
<program_body> "end" ";" .

<parameters_declaration> ::= <parameters_declaration> ";" <identifier_list> ":" <type>
                           | <parameters_declaration> ";" "var" <identifier_list> ":" <type>
```

```
                          | <identifier_list> ":" <type>
                          | "var" <identifier_list> ":" <type> .

<call_to_method> ::= <id> "(" <call_arguments> ")" .

<call_arguments> ::= <call_arguments> "," <expression>
                   | <expression>
                   | /* empty */ .

<argument> ::= <expression> .

<writeln_call> ::= "writeln" "(" <call_arguments> ")" .

<readln_call> ::= "readln" "(" <call_arguments> ")" .

<expression> ::= <id>
               | <integer>
               | <decimal>
               | <char>
               | <string>
               | "true"
               | "false"
               | <expression> "+" <expression>
               | <expression> "-" <expression>
               | <expression> "*" <expression>
               | <expression> "div" <expression>
               | <expression> "mod" <expression>
               | "-" <expression>
               | "not" <expression>
               | <expression> "and" <expression>
               | <expression> "or" <expression>
               | <expression> "and then" <expression>
               | <expression> "or else" <expression>
               | <expression> "=" <expression>
               | <expression> "<" <expression>
               | <expression> ">" <expression>
               | <expression> "<=" <expression>
               | <expression> ">=" <expression>
               | <expression> "<>" <expression>
               | <id> "[" <expression> "]"
               | <call_to_method>
               | "(" <expression> ")" .
```

INSTRUCCIONES PARA EJECUCIÓN

A continuación, se presentan los pasos necesarios para poner en funcionamiento el proyecto.

- Instalar la versión 21 de Java
- Mediante un gestor de archivos gráfico, abrir el archivo .jar
- También puede ser abierto a través de una terminal con el comando
  java -jar nombre_Del_Archivo.jar