PROYECTO 2

18.05.2022

Carlos Benjamín Pac Flores
Centro Universitario de Occidente (CUNOC)
Ingeniería en Ciencias y Sistemas
Mayo 2022

Introducción

El siguiente documento da presenta una descripción de la estructura del software de la aplicación de uso del intérprete CRl realizada con el analizador lexico/sintactico JISON escrito en JavaScript y Angular como fronted manejada por el lenguaje de TypeScript, de esta forma ofrecer al usuario una interfaz para poder trabajar con varias pestañas, subir archivos y ejecutar el código en el mismo entorno.

Plataforma de Desarrollo:

- Angular 13.3.3
- Visual Studio Code
- Jison (Analizador Lexico y sintactico)

Alcances Tecnicos:

Utilizando un aplicación web realizar el procesado del lenguaje interpretado CRL, en la aplicación web podemos tener acceso a crear múltiples archivos, los cuales puede ser usados por el lenguaje CRL, así también se incluye una consola integrada para ver los resultados de análisis y de ejecución del código, un apartado para poder visualizar los gráficos generados por el programa.

Requisitos De Software:

- Angular CLI
- Dependencias NPM (npm install)
- Ram recomendada 4 GB

Detalle del Sistema:

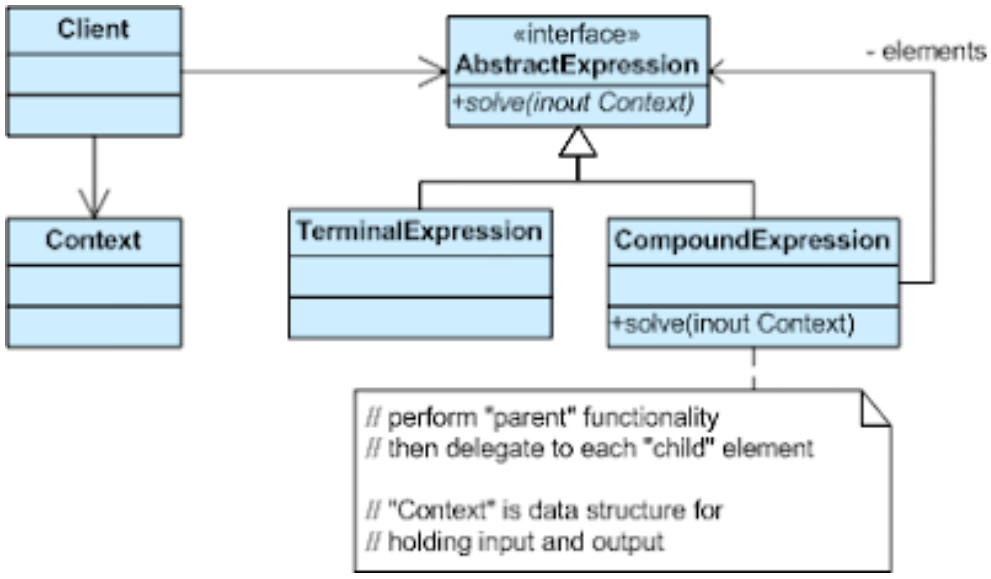
Patrón de Diseño

Debido a la forma con la cual esta planteado el lenguaje se necesita de poder ejecutar varias instrucciones dependiendo de su llamada y con esto nos vemos en la necesidad de realizar en algunos momentos otras llamadas desde la misma es decir podemos tener llamadas recursivas, por esa razón y ajustando a un modelo mucho más sencillo se hizo la implementación del “Patrón Interpreter” es cual es una abstracción en forma de objeto para su facil manipulacion.

Patron Interpreter:

El patrón Interpreter sugiere modelar el dominio con una gramática recursiva. Cada regla gramática puede ser compuesta (una regla que referencia a otras reglas) o terminal (una hoja en una estructura de tipo árbol).

Este patrón se basa en el recorrido recursivo del patrón Composite para interpretar las sentencias que se le solicita procesar.



Este modelo nos puede ayudar también a simplificar el funcionamiento del mismo, es decir que podemos realizar una similitud de instrucciones de nuestra tecnología fuente a lo que queremos resolver simplificando código y acciones dentro del mismo.

Estructura del analisis Lexico:

- Lenguaje CRL:

```
identificador ([a-zA-Z_$]([a-zA-Z_$]|[0-9])*)
comentSimple (("!!")([^\n]*))
comentMultip ((\'\'\')([^']*)(\'\'\'))

{comentSimple}      {/*Ingonorar un comentario simple*/}
{comentMultip}      {/*Ingonorar un comentario multiple*/}

\t+\n+              {return 'NUEVA_LINEA';}
\t+                 {return 'IDENTACION';}
\n+                 {return 'NUEVA_LINEA';}
\s                  {/*ingnorado*/}
[0-9]+"."[0-9]+     {return 'DECIMAL';}
[0-9]+              {return 'ENTERO';}
(\"[^"]*\")         {return 'CADENA';}
(\'[^"]\')          {return 'CARACTER';}
".crl"              {return 'EXTENCION_CRL';}

"*"             {return '*';}
"/"             {return '/';}
"%"             {return '%';}
"^"             {return '^';}
";"             {return ';';}
":"             {return ':';}
","             {return ',';}
"++"            {return '++';}
"--"            {return '--';}
"-"             {return '-';}
"+"             {return '+';}

"<="            {return '<=';}
">="            {return '>=';}
"=="            {return '==';}
"!="            {return '!=';}
"||"            {return '||';}
"|&"            {return '|&';}
"&&"            {return '&&';}
"!"             {return '!';}
"<"             {return '<';}
```

```
">"             {return '>';}
"="             {return '=';}
"~"             {return '~';}

"("             {return '(';}
")"             {return ')';}

"Double"        {return 'DOUBLE';}
"Boolean"       {return 'BOOLEAN';}
"String"        {return 'STRING';}
"Int"           {return 'INT';}
"Char"          {return 'CHAR';}
"Void"          {return 'VOID';}
"true"          {return 'TRUE';}
"false"         {return 'FALSE';}
"Si"            {return 'SI';}
"Sino"          {return 'SINO';}
"Para"          {return 'PARA';}
"Mientras"      {return 'MIENTRAS';}
"Detener"       {return 'DETENER';}
"Continuar"     {return 'CONTINUAR';}
"Retorno"       {return 'RETORNO';}
"Mostrar"       {return 'MOSTRAR';}
"Importar"      {return 'IMPORTAR';}
"Incerteza"     {return 'INCERTEZA';}
"DibujarAST"    {return 'DIBUJAR_AST';}
"DibujarEXP"    {return 'DIBUJAR_EXP';}
"DibujarTS"     {return 'DIBUJAR_TS';}
"Principal"     {return 'PRINCIPAL';}
{identificador} {return 'ID';}
<<EOF>>         {return 'EOF';}

.               {Marca un error léxico en el código del programa}
```

Estructura del Análisis Sintáctico:

- Lenguaje CRL

Símbolos Terminales:

```
NUEVA_LINEA, IDENTACION, DECIMAL, ENTERO, CADENA, CARACTER, EXTENCION_CRL,
*, /, %, ^, ;, :, ,, ++, --, -, +, <=, >=, ==, !=, ||, |&, &&, !, <, >, =,
~, (, ), DOUBLE, BOOLEAN, STRING, INT, CHAR, VOID, TRUE, FALSE, SI, SINO,
```

PARA, MIENTRAS, DETENER, CONTINUAR, RETORNO, MOSTRAR, IMPORTAR, INCERTEZA, DIBUJAR_AST, DIBUJAR_EXP, DIBUJAR_TS, PRINCIPAL, ID, EOF

Símbolos No Terminales:

Init, inicioCode, instruccionesEncabezado, importacion, defIncerteza, intrucciones, instruction, instructionGlobal, funcionDibujarTs, funcionDibujarExp, funcionDibujarAST, funcionMostrar, sentenciaContinuar, sentenciaDetener, sentenciaMientras, sentenciaPara, opPara, sentenciaSi, instruccionRetorno, llamarFuncion, parametrosEnviar, instruccionFuncionMetodo, parametros, intruccionAsignar, intruccionDeclarar, tipoDato, listaIds, exprecion, f

Reglas De Producción:

Reglas de Precedencia

```
izquierda '+','-'
izquierda '*','/','%'
derecha   '^'
izquierda '<','>','<=','>='
izquierda '==','!=','~'
izquierda '||'
izquierda '|&'
izquierda '&&'
izquierda '!'
izquierda UMINUS
```

```
Init -> inicioCode EOF

inicioCode -> instrucciones

intruccionesEncabezado -> importacion
intruccionesEncabezado -> defIncerteza


importacion -> IMPORTAR ID EXTENCION_CRL

defIncerteza -> INCERTEZA DECIMAL

instrucciones -> instrucciones instruction
instrucciones -> instruction

instruction -> instructionGlobal NUEVA_LINEA
instruction -> instruccionFuncionMetodo NUEVA_LINEA
instruction -> intruccionesEncabezado NUEVA_LINEA
```

```
instruction -> VOID PRINCIPAL '(' ')' ':' NUEVA_LINEA
instruction -> NUEVA_LINEA
instruction -> IDENTACION NUEVA_LINEA

instructionGlobal -> instruccionDeclarar
instructionGlobal -> instruccionAsignar
instructionGlobal -> llamarFuncion
instructionGlobal -> instruccionRetorno
instructionGlobal -> sentenciaSi
instructionGlobal -> sentenciaPara
instructionGlobal -> sentenciaMientras
instructionGlobal -> sentenciaDetener
instructionGlobal -> sentenciaContinuar
instructionGlobal -> funcionMostrar
instructionGlobal -> funcionDibujarAST
instructionGlobal -> funcionDibujarExp
instructionGlobal -> funcionDibujarTs

funcionDibujarTs -> IDENTACION DIBUJAR_TS '('')'

funcionDibujarExp -> IDENTACION DIBUJAR_EXP '(' exprecion ')'

funcionDibujarAST -> IDENTACION DIBUJAR_AST '(' identificador ')'

funcionMostrar -> IDENTACION MOSTRAR '(' exprecion ',' parametrosEnviar ')'
funcionMostrar -> IDENTACION MOSTRAR '(' exprecion ')'

sentenciaContinuar -> IDENTACION CONTINUAR

sentenciaDetener -> IDENTACION DETENER

sentenciaMientras -> IDENTACION MIENTRAS '(' exprecion ')' ':'

sentenciaPara -> IDENTACION PARA '('INT ID '=' exprecion ';' exprecion ';' opPara
')' ':'

opPara -> '++'
opPara -> '--'

sentenciaSi -> IDENTACION SI '(' exprecion ')' ':'
sentenciaSi -> IDENTACION SINO ':'

instruccionRetorno -> IDENTACION RETORNO exprecion

llamarFuncion -> IDENTACION ID '(' parametrosEnviar ')'
llamarFuncion -> IDENTACION ID '(' ')'
```

```
parametrosEnviar -> parametrosEnviar ',' exprecion
parametrosEnviar -> exprecion

instruccionFuncionMetodo -> tipoDato ID '(' parametros ')' ':'
instruccionFuncionMetodo -> tipoDato ID '(' ')' ':'

parametros -> parametros ',' tipoDato ID
parametros -> tipoDato ID

instruccionAsignar -> ID '=' exprecion
instruccionAsignar -> IDENTACION ID '=' exprecion

instruccionDeclarar -> IDENTACION tipoDato listaIds
instruccionDeclarar -> tipoDato listaIds

tipoDato -> INT
tipoDato -> STRING
tipoDato -> CHAR
tipoDato -> DOUBLE
tipoDato -> BOOLEAN
tipoDato -> VOID

listaIds -> listaIds ',' ID
listaIds -> listaIds ',' ID '=' exprecion
listaIds -> ID
listaIds -> ID '=' exprecion

exprecion -> '-' exprecion %prec UMINUS
exprecion -> exprecion '+' exprecion
exprecion -> exprecion '-' exprecion
exprecion -> exprecion '/' exprecion
exprecion -> exprecion '^' exprecion
exprecion -> exprecion '*' exprecion
exprecion -> exprecion '%' exprecion
exprecion -> exprecion '>' exprecion
exprecion -> exprecion '<' exprecion
exprecion -> exprecion '>=' exprecion
exprecion -> exprecion '<=' exprecion
exprecion -> exprecion '!=' exprecion
exprecion -> exprecion '==' exprecion
exprecion -> exprecion '~' exprecion
exprecion -> exprecion '||' exprecion
exprecion -> exprecion '|&' exprecion
exprecion -> exprecion '&&' exprecion
exprecion -> '!' exprecion
exprecion -> f
```

```
f -> '(' exprecion ')'
f -> DECIMAL
f -> ENTERO
f -> CADENA
f -> CARACTER
f -> TRUE
f -> FALSE
f -> ID
f -> ID '(' ')'
f -> ID '(' parametrosEnviar ')'
```

Código de Ejemplo CRL:

```
!!Aqui estoy declarando una importacion y una incerteza.
Importar aritmeticas.crl
Incerteza 0.5

!!Declaro algunas globales
Int Estado=(0+10-10)^1
String palabra, palabra1="Hola Mundo"
!!palabra1="Adios Mundo"


Void Principal():
      Saludar(palabra, true, true)
      Saludar(palabra, true, false)
      Saludar(palabra, false, false)
      Int resultado=factorial(5)*true
      Int resultado2=factorial(1)*true
      Mostrar("El factorial de: {0} es: {1} y el factorial de: {2} por 0 es:
{3}",5,resultado,1,resultado2)
      Imprimir(2,2)
      DibujarEXP((60+5)*num1)
      Imprimir2(2,2)
      Contar(10)
      DibujarAST(Graficar)
```

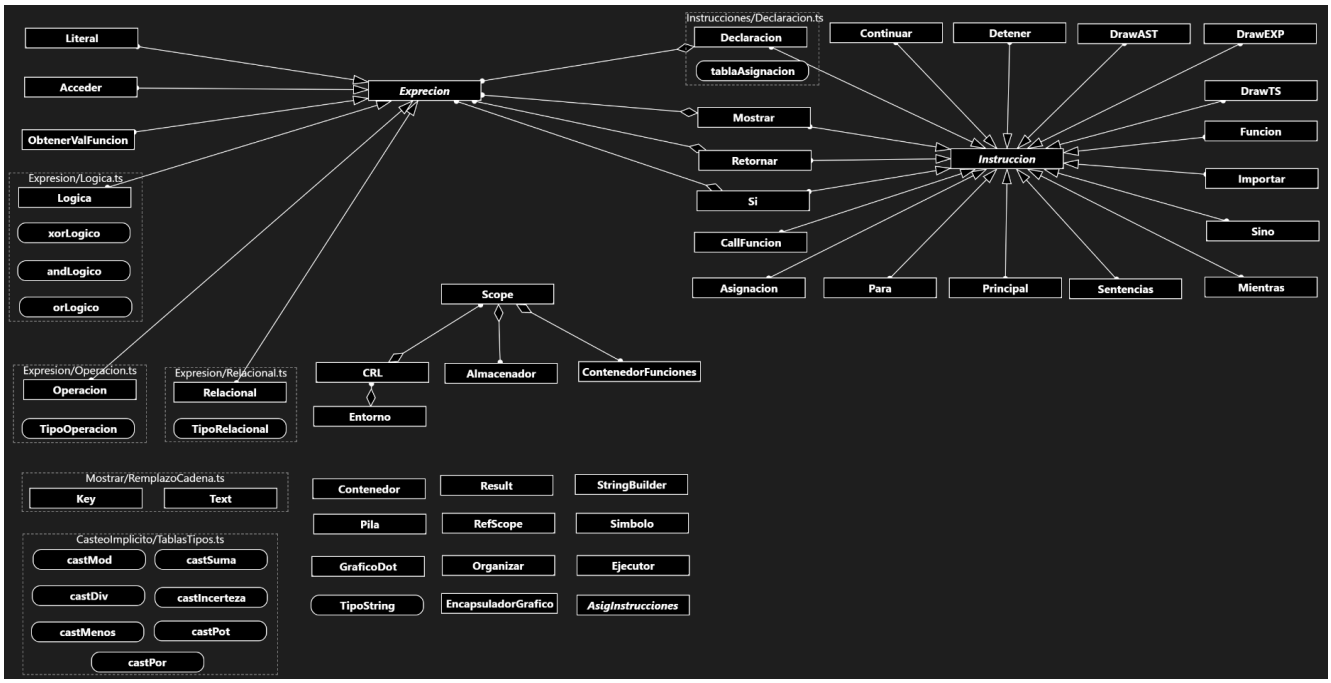
```
!!Aqui estoy declarando una incerteza.
Incerteza 1.5
```

```
Int factorial(Int n):
      Int num = 1
      Mientras (n >= 1):
            num = num * n
            n = n - 1
      Retorno num


Void Saludar(String mensaje, Boolean tipo, Boolean tipo2):
      Si (tipo && tipo1):
            Mostrar ("Dar saludo")
            Mostrar (Mensaje)
      Sino:
            Si(tipo |& tipo1):
                  Mostrar ("Sigue en el programa")
            Sino:
                  Mostrar ("Dar despedida")
                  Mostrar (Mensaje)
            Mostrar ("Se acabo el sino")
      Mostrar ("Se acabo la funcion")



Void imprimr(Int a, Int b):
      Para (Int i=1; i<a; ++):
            Para (Int j=1; j<b;++):
                  Si(j~3):
                        Mostrar ("Matriz x={0}, y={1}",i,j)

Void imprimr2(Int a, Int b):
      Para (Int i=1; i<a; ++):
            Para (Int j=1; j<b;++):
                  Si(j~3):
                        Continuar
                  Mostrar ("Matriz x={0}, y={1}",i,j)

Void Contar(Int a):
      Para (Int i=0; i<a;++):
            Si (i>=5):
                  Detener
            Mostrar ("numero: {0}", i)
```

```
Double Graficar(Double a):
      Si (a>10):
            Retorno 1
      Retorno a
```

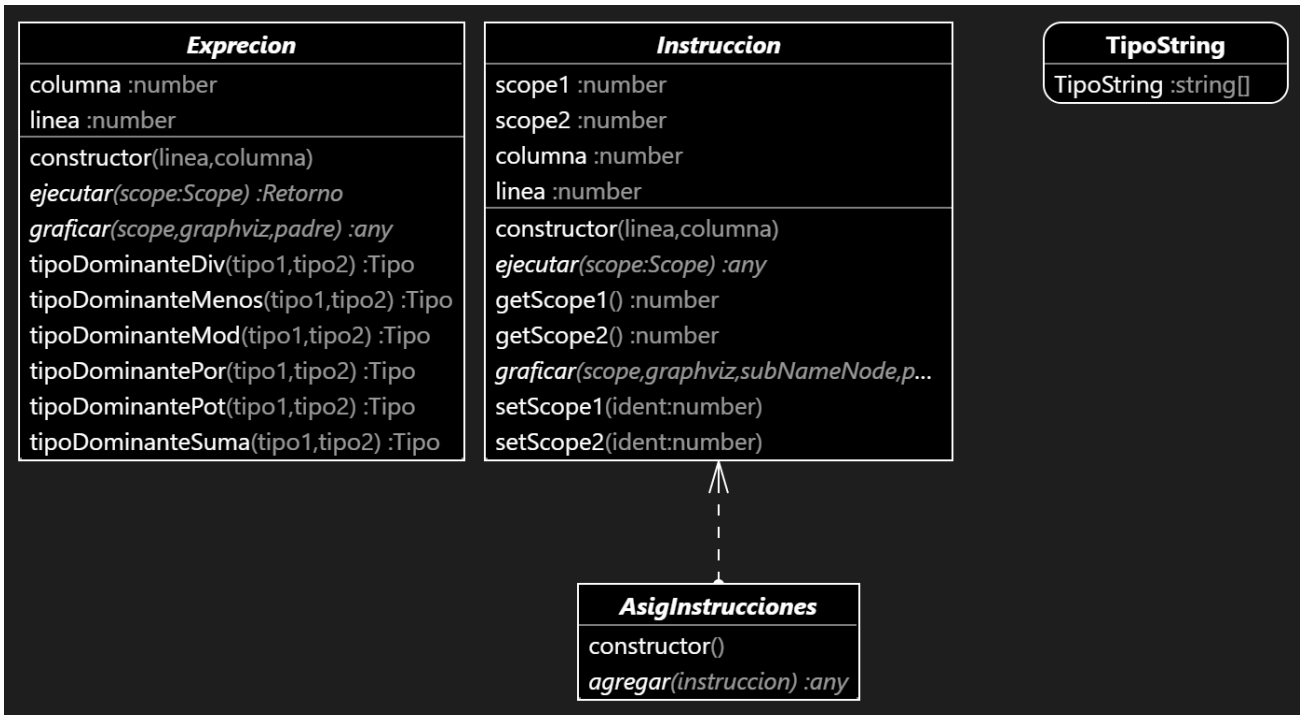
Estructura del Proyecto:

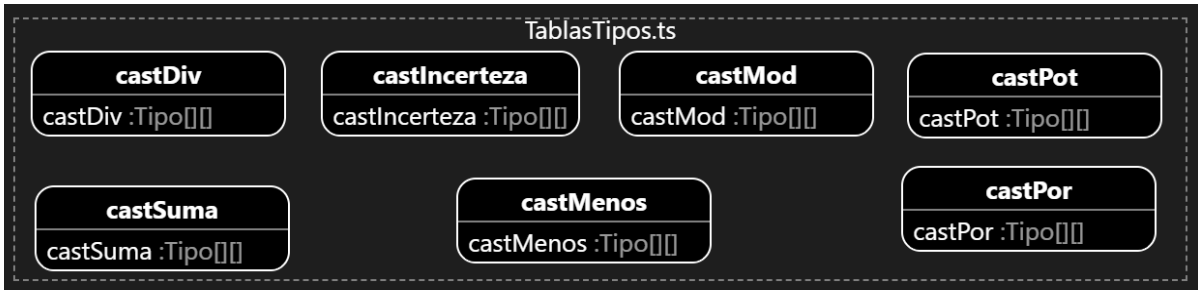
Diagrama de Clases:

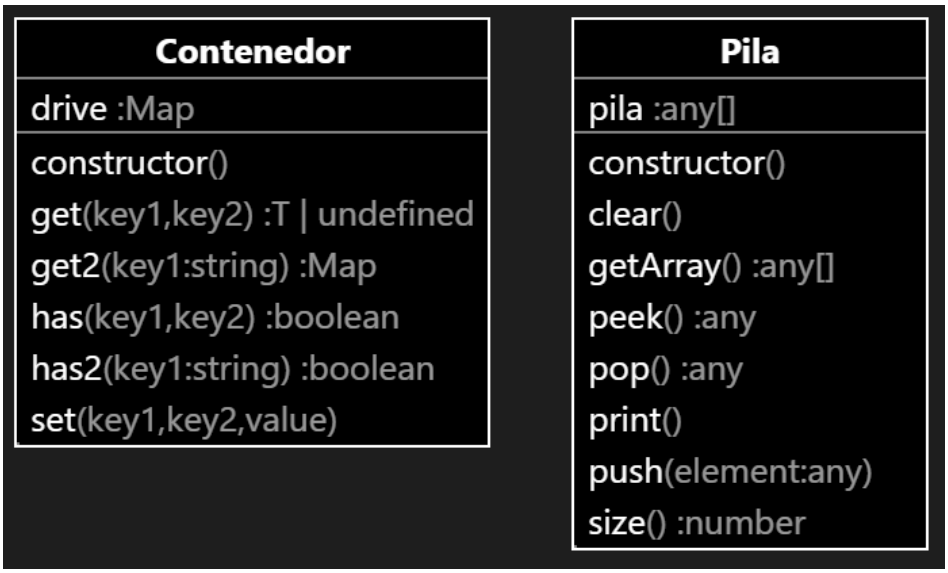


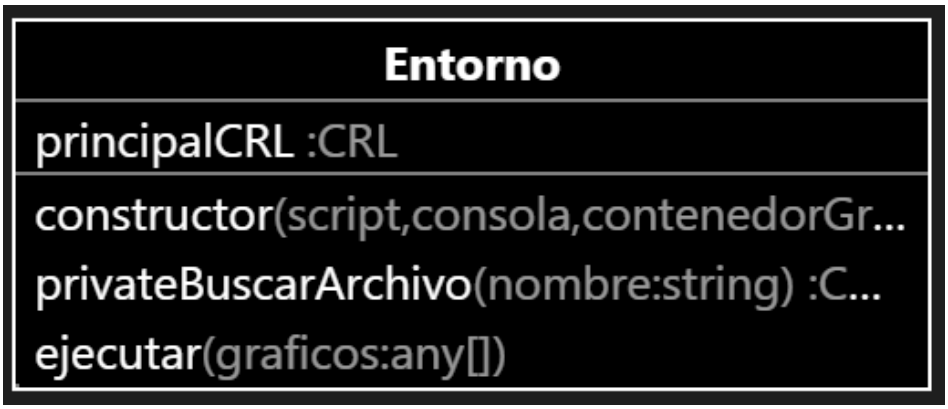
Es presentado de esta manera ya que su descripción general completa no permitirá visualizarlos de una manera correcta, por ende adelante de esta sección se adjunta su descripción completa.

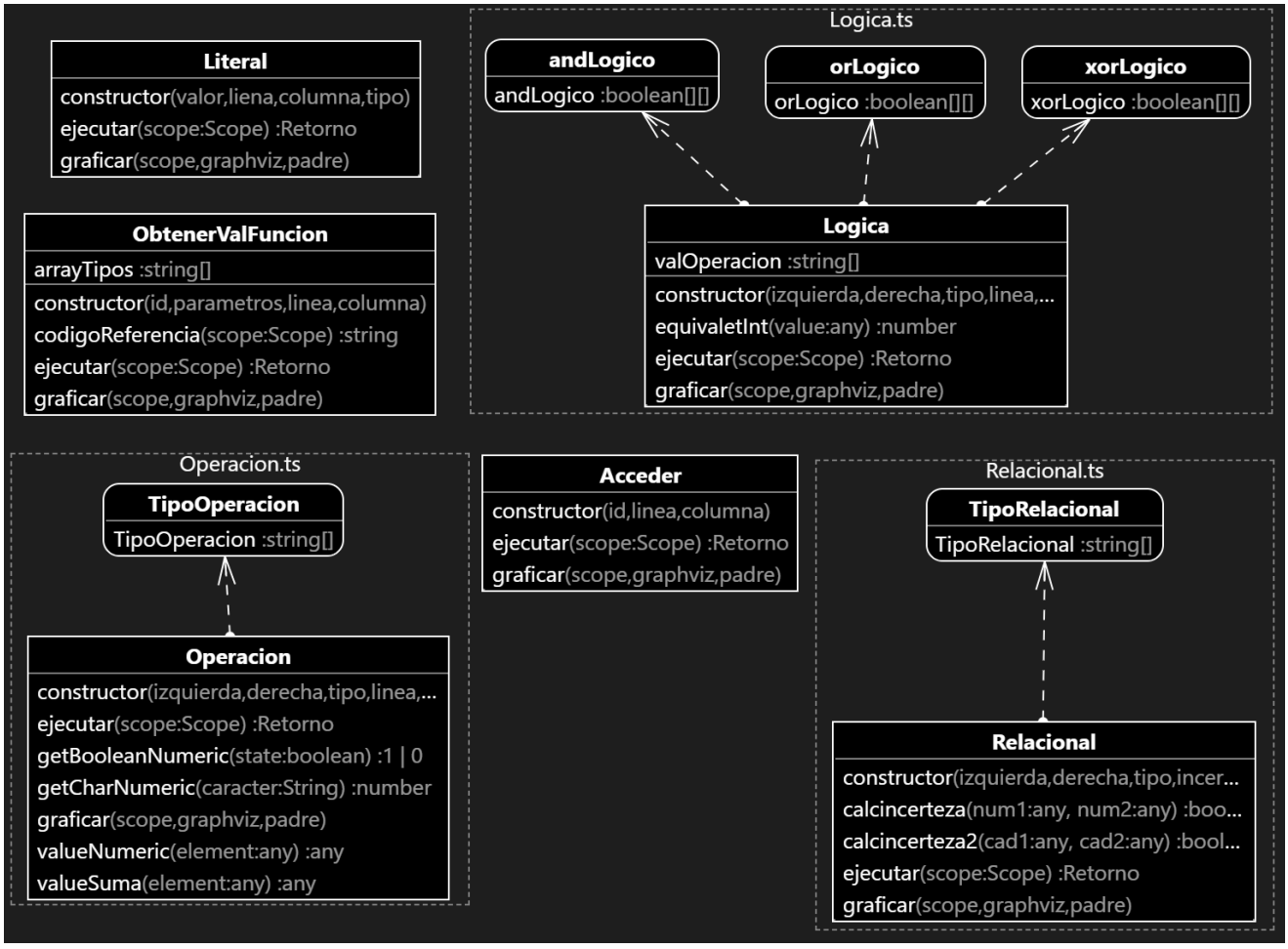
Contenido de las clases utilizadas:


Expresion
columna :number
linea :number
constructor(linea,columna)
ejecutar(scope:Scope) :Retorno
graficar(scope,graphviz,padre) :any
tipoDominanteDiv(tipo1,tipo2) :Tipo
tipoDominanteMenos(tipo1,tipo2) :Tipo
tipoDominanteMod(tipo1,tipo2) :Tipo
tipoDominantePor(tipo1,tipo2) :Tipo
tipoDominantePot(tipo1,tipo2) :Tipo
tipoDominanteSuma(tipo1,tipo2) :Tipo
Instruccion
scope1 :number
scope2 :number
columna :number
linea :number
constructor(linea,columna)
ejecutar(scope:Scope) :any
getScope1() :number
getScope2() :number
graficar(scope,graphviz,subNameNode,p...
setScope1(ident:number)
setScope2(ident:number)
TipoString
TipoString :string[]
AsigInstrucciones
constructor()
agregar(instruccion) :any

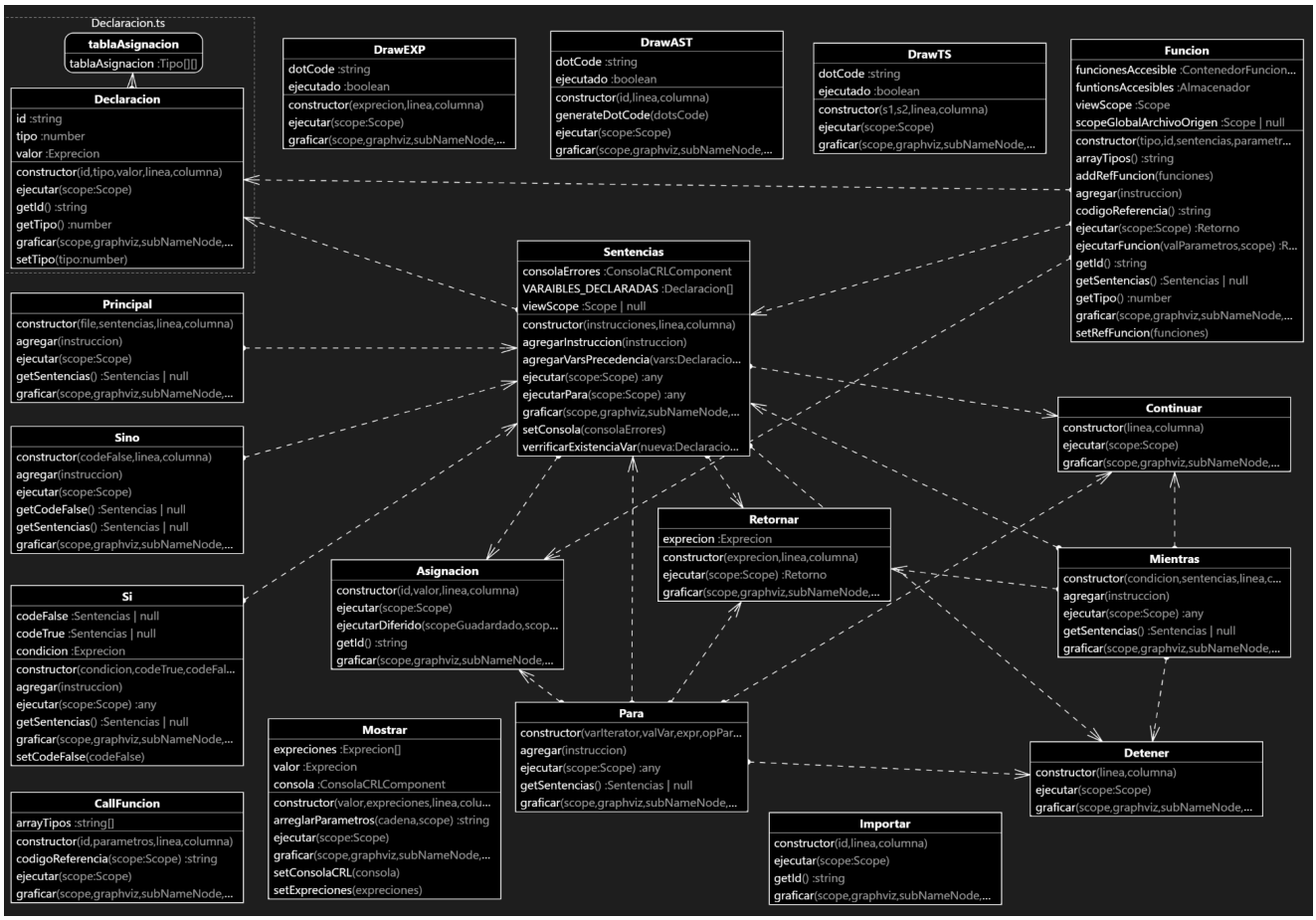

TablasTipos.ts
castDiv
castDiv :Tipo[][]
castIncerteza
castIncerteza :Tipo[][]
castMod
castMod :Tipo[][]
castPot
castPot :Tipo[][]
castSuma
castSuma :Tipo[][]
castMenos
castMenos :Tipo[][]
castPor
castPor :Tipo[][]

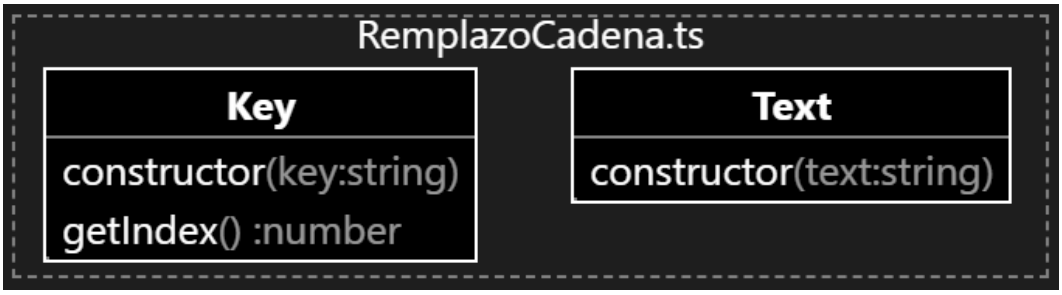

Contenedor
drive :Map
constructor()
get(key1,key2) :T | undefined
get2(key1:string) :Map
has(key1,key2) :boolean
has2(key1:string) :boolean
set(key1,key2,value)
Pila
pila :any[]
constructor()
clear()
getArray() :any[]
peek() :any
pop() :any
print()
push(element:any)
size() :number

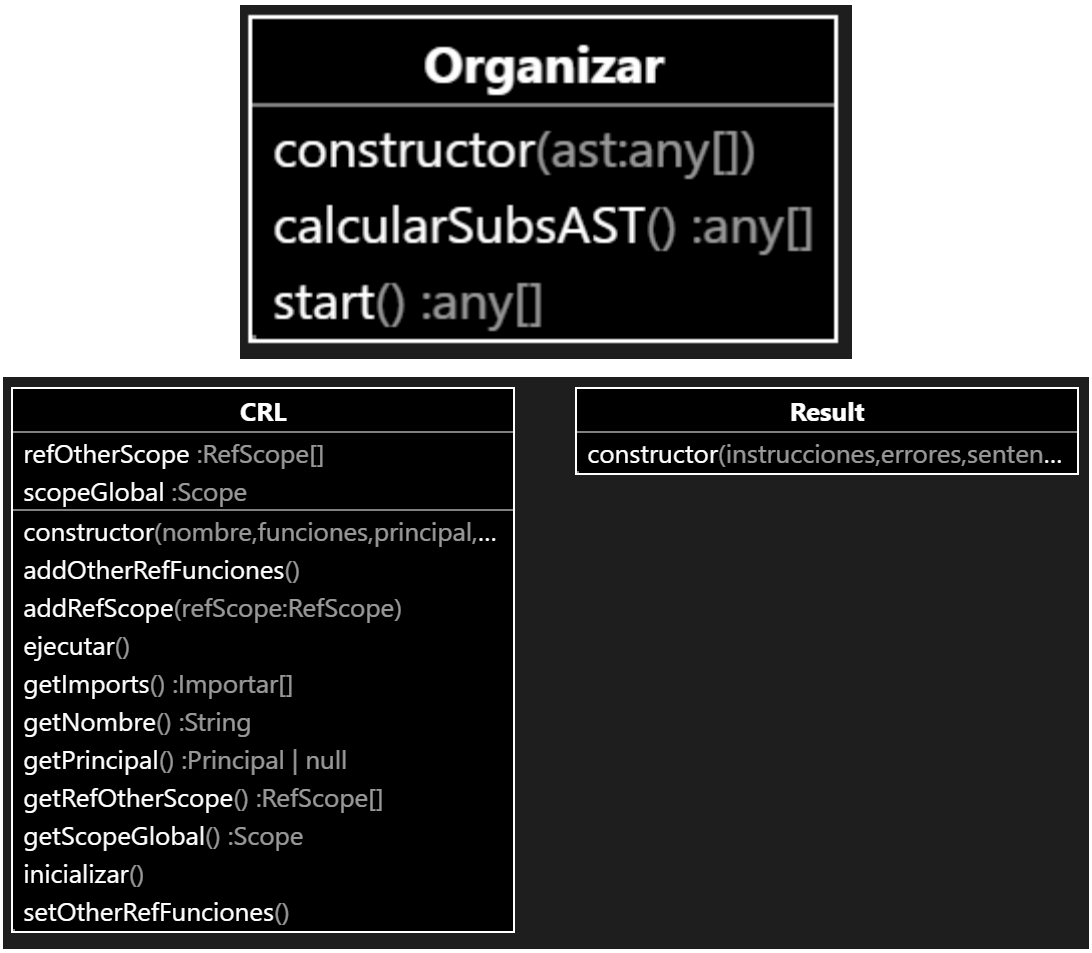

Entorno
principalCRL :CRL
constructor(script,consola,contenedorGr...
privateBuscarArchivo(nombre:string) :C...
ejecutar(graficos:any[])

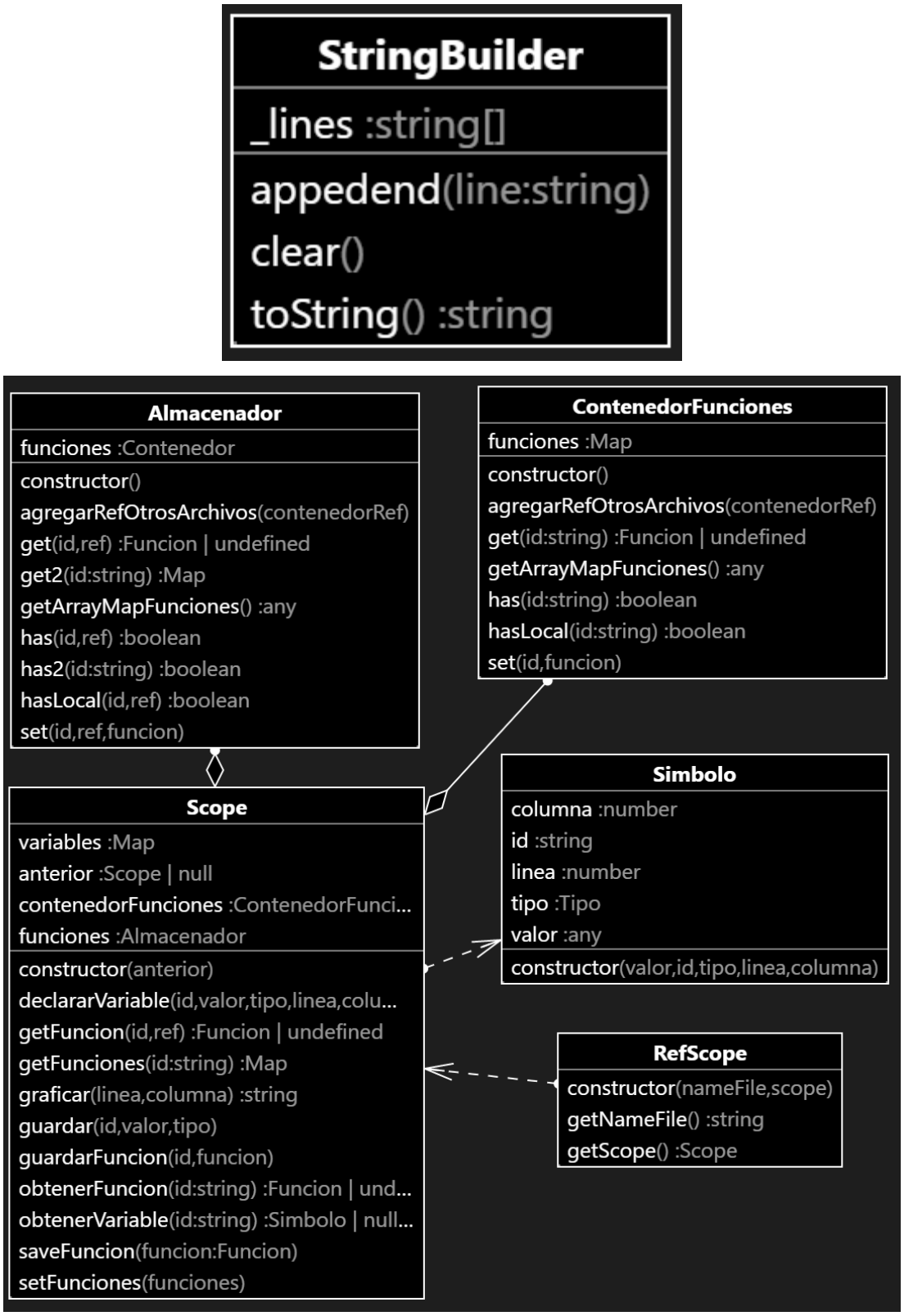

Logica.ts
Literal
constructor(valor,liena,columna,tipo)
ejecutar(scope:Scope) :Retorno
graficar(scope,graphviz,padre)
andLogico
andLogico :boolean[][]
orLogico
orLogico :boolean[][]
xorLogico
xorLogico :boolean[][]
ObtenerValFuncion
arrayTipos :string[]
constructor(id,parametros,linea,columna)
codigoReferencia(scope:Scope) :string
ejecutar(scope:Scope) :Retorno
graficar(scope,graphviz,padre)
Logica
valOperacion :string[]
constructor(izquierda,derecha,tipo,linea,...
equivaletInt(value:any) :number
ejecutar(scope:Scope) :Retorno
graficar(scope,graphviz,padre)
Operacion.ts
TipoOperacion
TipoOperacion :string[]
Operacion
constructor(izquierda,derecha,tipo,linea,...
ejecutar(scope:Scope) :Retorno
getBooleanNumeric(state:boolean) :1 | 0
getCharNumeric(caracter:String) :number
graficar(scope,graphviz,padre)
valueNumeric(element:any) :any
valueSuma(element:any) :any
Acceder
constructor(id,linea,columna)
ejecutar(scope:Scope) :Retorno
graficar(scope,graphviz,padre)
Relacional.ts
TipoRelacional
TipoRelacional :string[]
Relacional
constructor(izquierda,derecha,tipo,incer...
calcincerteza(num1:any, num2:any) :boo...
calcincerteza2(cad1:any, cad2:any) :bool...
ejecutar(scope:Scope) :Retorno
graficar(scope,graphviz,padre)

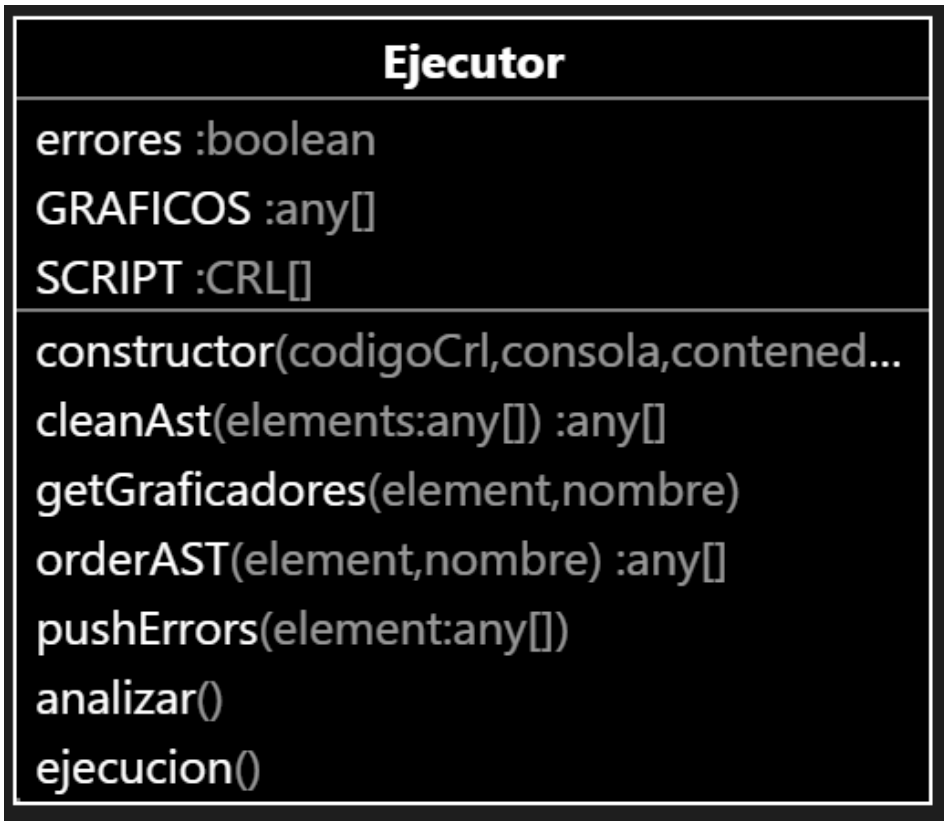

EncapsuladorGrafico
constructor(obj,file)
GraficoDot
declaraciones :string[]
relaciones :string[]


Declaracion.ts
tablaAsignacion
tablaAsignacion :Tipo[][]
Declaracion
id :string
tipo :number
valor :Exprecion
constructor(id,tipo,valor,linea,columna)
ejecutar(scope:Scope)
getId() :string
getTipo() :number
graficar(scope,graphviz,subNameNode,...
setTipo(tipo:number)
DrawEXP
dotCode :string
ejecutado :boolean
constructor(exprecion,linea,columna)
ejecutar(scope:Scope)
graficar(scope,graphviz,subNameNode,...
DrawAST
dotCode :string
ejecutado :boolean
constructor(id,linea,columna)
generateDotCode(dotsCode)
ejecutar(scope:Scope)
graficar(scope,graphviz,subNameNode,...
DrawTS
dotCode :string
ejecutado :boolean
constructor(s1,s2,linea,columna)
ejecutar(scope:Scope)
graficar(scope,graphviz,subNameNode,...
Funcion
funcionesAccesible :ContenedorFuncion...
funtionsAccesibles :Almacenador
viewScope :Scope
scopeGlobalArchivoOrigen :Scope | null
constructor(tipo,id,sentencias,parametr...
arrayTipos() :string
addRefFuncion(funciones)
agregar(instruccion)
codigoReferencia() :string
ejecutar(scope:Scope) :Retorno
ejecutarFuncion(valParametros,scope) :R...
getId() :string
getSentencias() :Sentencias | null
getTipo() :number
graficar(scope,graphviz,subNameNode,...
setRefFuncion(funciones)
Principal
constructor(file,sentencias,linea,columna)
agregar(instruccion)
ejecutar(scope:Scope)
getSentencias() :Sentencias | null
graficar(scope,graphviz,subNameNode,...
Sentencias
consolaErrores :ConsolaCRLComponent
VARAIBLES_DECLARADAS :Declaracion[]
viewScope :Scope | null
constructor(instrucciones,linea,columna)
agregarInstruccion(instruccion)
agregarVarsPrecedencia(vars:Declaracio...
ejecutar(scope:Scope) :any
ejecutarPara(scope:Scope) :any
graficar(scope,graphviz,subNameNode,...
setConsola(consolaErrores)
verrificarExistenciaVar(nueva:Declaracio...
Continuar
constructor(linea,columna)
ejecutar(scope:Scope)
graficar(scope,graphviz,subNameNode,...
Sino
constructor(codeFalse,linea,columna)
agregar(instruccion)
ejecutar(scope:Scope)
getCodeFalse() :Sentencias | null
getSentencias() :Sentencias | null
graficar(scope,graphviz,subNameNode,...
Retornar
exprecion :Exprecion
constructor(exprecion,linea,columna)
ejecutar(scope:Scope) :Retorno
graficar(scope,graphviz,subNameNode,...
Asignacion
constructor(id,valor,linea,columna)
ejecutar(scope:Scope)
ejecutarDiferido(scopeGuadardado,scop...
getId() :string
graficar(scope,graphviz,subNameNode,...
Mientras
constructor(condicion,sentencias,linea,c...
agregar(instruccion)
ejecutar(scope:Scope) :any
getSentencias() :Sentencias | null
graficar(scope,graphviz,subNameNode,...
Si
codeFalse :Sentencias | null
codeTrue :Sentencias | null
condicion :Exprecion
constructor(condicion,codeTrue,codeFal...
agregar(instruccion)
ejecutar(scope:Scope) :any
getSentencias() :Sentencias | null
graficar(scope,graphviz,subNameNode,...
setCodeFalse(codeFalse)
Mostrar
expreciones :Exprecion[]
valor :Exprecion
consola :ConsolaCRLComponent
constructor(valor,expreciones,linea,colu...
arreglarParametros(cadena,scope) :string
ejecutar(scope:Scope)
graficar(scope,graphviz,subNameNode,...
setConsolaCRL(consola)
setExpreciones(expreciones)
Para
constructor(varIterator,valVar,expr,opPar...
agregar(instruccion)
ejecutar(scope:Scope) :any
getSentencias() :Sentencias | null
graficar(scope,graphviz,subNameNode,...
Detener
constructor(linea,columna)
ejecutar(scope:Scope)
graficar(scope,graphviz,subNameNode,...
CallFuncion
arrayTipos :string[]
constructor(id,parametros,linea,columna)
codigoReferencia(scope:Scope) :string
ejecutar(scope:Scope)
graficar(scope,graphviz,subNameNode,...
Importar
constructor(id,linea,columna)
ejecutar(scope:Scope)
getId() :string
graficar(scope,graphviz,subNameNode,...


RemplazoCadena.ts
Key
constructor(key:string)
getIndex() :number
Text
constructor(text:string)


Organizar
constructor(ast:any[])
calcularSubsAST() :any[]
start() :any[]
CRL
refOtherScope :RefScope[]
scopeGlobal :Scope
constructor(nombre,funciones,principal,...
addOtherRefFunciones()
addRefScope(refScope:RefScope)
ejecutar()
getImports() :Importar[]
getNombre() :String
getPrincipal() :Principal | null
getRefOtherScope() :RefScope[]
getScopeGlobal() :Scope
inicializar()
setOtherRefFunciones()
Result
constructor(instrucciones,errores,senten...


StringBuilder
_lines :string[]
appedend(line:string)
clear()
toString() :string
Almacenador
funciones :Contenedor
constructor()
agregarRefOtrosArchivos(contenedorRef)
get(id,ref) :Funcion | undefined
get2(id:string) :Map
getArrayMapFunciones() :any
has(id,ref) :boolean
has2(id:string) :boolean
hasLocal(id,ref) :boolean
set(id,ref,funcion)
ContenedorFunciones
funciones :Map
constructor()
agregarRefOtrosArchivos(contenedorRef)
get(id:string) :Funcion | undefined
getArrayMapFunciones() :any
has(id:string) :boolean
hasLocal(id:string) :boolean
set(id,funcion)
Scope
variables :Map
anterior :Scope | null
contenedorFunciones :ContenedorFunci...
funciones :Almacenador
constructor(anterior)
declararVariable(id,valor,tipo,linea,colu...
getFuncion(id,ref) :Funcion | undefined
getFunciones(id:string) :Map
graficar(linea,columna) :string
guardar(id,valor,tipo)
guardarFuncion(id,funcion)
obtenerFuncion(id:string) :Funcion | und...
obtenerVariable(id:string) :Simbolo | null...
saveFuncion(funcion:Funcion)
setFunciones(funciones)
Simbolo
columna :number
id :string
linea :number
tipo :Tipo
valor :any
constructor(valor,id,tipo,linea,columna)
RefScope
constructor(nameFile,scope)
getNameFile() :string
getScope() :Scope


Ejecutor
errores :boolean
GRAFICOS :any[]
SCRIPT :CRL[]
constructor(codigoCrl,consola,contened...
cleanAst(elements:any[]) :any[]
getGraficadores(element,nombre)
orderAST(element,nombre) :any[]
pushErrors(element:any[])
analizar()
ejecucion()