Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Compiladores e Intérpretes

Manual Técnico

García Amado Andrés
LU: 89522

Índice

Índice **1**

Introducción: **3**

Alfabeto de entrada: **4**

Tokens reconocidos por el analizador léxico: **4**

Palabras Reservadas: **5**

Estructura del Analizador: **6**

Errores léxicos y casos de test: **7**
Errores léxicos: 7
Casos de test: 7

Análisis y Transformación de la gramática: **8**

Eliminación de símbolos de BNF extendida: **9**

Eliminación de Recursión a Izquierda: **12**

Factorización: **13**

Estructura del analizador sintáctico: **16**

Esquema de Traducción: **16**

Diagrama de clases: **26**

Explicación de los Métodos de la Tabla de Símbolos: **30**

Explicación de los chequeos que realiza el compilador **31**
Chequeo de declaraciones de una clase: 31
Chequeo de declaraciones de un método: 31
Chequeo de declaraciones de un parámetro: 31
Chequeo en sentencias: 32
Chequeo en expresiones: 32

Cálculo de offsets y etiquetas **33**
Generacion de las etiquetas de las unidades: 33
Calculo de Offset de las variables de instancia: 33
Calculo de los offsets de parámetros: 33
Calculo de los offset de variables locales: 33

Calculo de los offsets de unidades: 33

Generación de Código: **34**

Introducción:

En este informe contiene todo lo referido a la información técnica de las etapas que llevan a la creación de un compilador, desde el comienzo viendo la gramática que utiliza MiniJava, como se realizan los controles en el analizador sintáctico, la creación de EDT y como se crea el árbol sintáctico abstracto (AST) y por último la generación del código. Además, se explicará las decisiones de diseño y su ejecución. Por último, se mostrarán algunos casos de Test, y se indicarán el resultado obtenido.

Alfabeto de entrada:

El alfabeto de entrada utilizado es el ASCII. Corresponde al conjunto de todos los caracteres que forman el conjunto de caracteres validos para crear palabras válidas para el lenguaje de MINIJAVA.

Alfabeto Σ= ASCII.

Tokens reconocidos por el analizador léxico:

Token	Expresión Regular
idClase	(A..Z) ((A..Z) + (a..z) + (0..9) + (_))*
idMetVal	(a..z) ((A..Z) + (a..z) + (0..9) + (_))*
entero	(0..9) | (0..9)*
caracter	‘(/)Σ’
tString	“ Σ* ”
null	null
parentesisAbre	(
parentesisCierra	)
puntoYComa	;
coma	,
punto	.
corcheteAbre	[
corcheteCierra	]
mayor	>
menor	<
not	!
igual	==
mayorIgual	>=

menorIgual	<=
asignacion	=
suma	+
resta	-
multiplicacion	*
division	/
and	&&
or	||

Palabras Reservadas:

Token	Expresión regular
class	class
extends	extends
static	static
dynamic	dynamic
String	String
boolean	boolean
char	char
int	int
public	public
private	private
void	void
null	null
if	if
else	else

while	while
return	return
this	this
new	new
true	true
false	false

Además de estos tokens, el analizador léxico reconoce el token de fin de archivo (EOF), para saber cuándo el análisis del archivo finalizó.

Estructura del Analizador:

- Clase Main:
  - Recibe un archivo.
  - Se encarga de iniciar el analizador.
  - Le pasa el archivo.
  - Le pide al analizador tokens.
  - imprime en pantalla cada token.
- Clase Analizador:
  - Abre el archivo.
  - Lee el archivo una sola vez y devuelve un token o un error en caso de encontrar uno.
  - utiliza el método getToken() para cumplir la función anterior.
- Token:
  - Es la clase que estructura al token.
  - Contiene el nombre, lexema, el número de columna y el número de fila.
- Excepciones:
  - ExAndMalFormado.
  - ExCaracterInvalido.
  - ExCaracterMalFormado.
  - ExComentarioMalFormado.
  - ExNumeroMalFormado.
  - ExOrMalFormado.
  - ExStringMalFormado.

Errores léxicos y casos de test:

Errores léxicos:

ExAndMalFormado: En el caso de que se lea un '&' y no haya otro '&'.
ExCaracterInvalido: Cuando la palabra comienza con un carácter inválido.
ExCaracterMalFormado: En el caso de que no se respete:

-'x' donde x es cualquier carácter excepto una barra invertida (\), salto de linea, o comilla simple (').
-'\x' donde x es cualquier carácter excepto n o t. El valor del literal es el valor del carácter x.
-'\t'- es el valor del literal es el valor del carácter tab.
-'\n'. Es valor del literal es el valor del carácter salto de línea.

ExComentarioMalFormado: Cuando un comentario no tiene el formato correspondiente.

-/*Comentario*/, en caso de ser un comentario multilínea.
-// Comentario, en el caso de ser un comentario de línea.

ExIdMalFormado: Cuando un Id no comienza con una mayúscula o cuando contiene un carácter que no es mayuscula, minuscula, numero, o guión bajo.
ExNumeroMalFormado: Cuando un número comienza con un dígito (entre 0..9) pero los caracteres seguidos no son dígitos.
ExOrMalFormado: En el caso de que se lea un '|' pero no es seguido por otro '|'.
ExStringMalFormado: Cuando el string contiene un carácter no válido.

Casos de test:

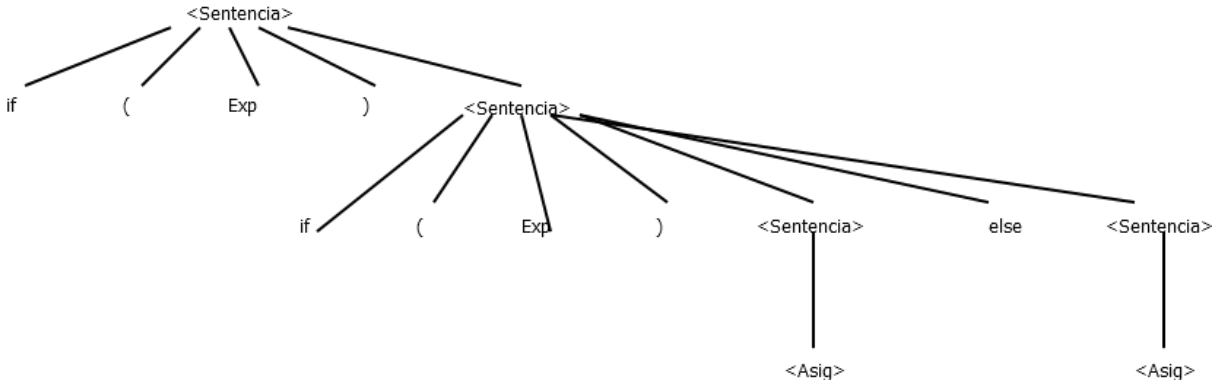
- Prueba 1: Verifica que se crea un comentario mal escrito.
- Prueba 2: Verifica que se crea un comentario mal cerrado.
- Prueba 3: Verifica el correcto funcionamiento del comentario multilínea.
- Prueba 4: Verifica el correcto funcionamiento del comentario de línea.
- Prueba 5: Verifica que un caracter no puede ser vacío.
- Prueba 6: Verifica un caracter no este mal cerrado.
- Prueba 7: Verifica que un string no este mal cerrado.
- Prueba 8: Verifica que un idClase no está mal escrito.
- Prueba 9: Verifica que un idMetVal no está mal escrito.
- Prueba 10: Verifica que un entero no este mal escrito.
- Prueba 11: Verifica que un OR no esta mal escrito.
- Prueba 12: Verifica que un AND no está mal escrito.
- Prueba 13: Test válido, la ejecución del mismo tiene que dar un resultado positivo.
- Prueba 14: Verifica que los tString con salto de línea (\n) en su interior sea interpretado correctamente al momento de mostrarlos en la tabla.

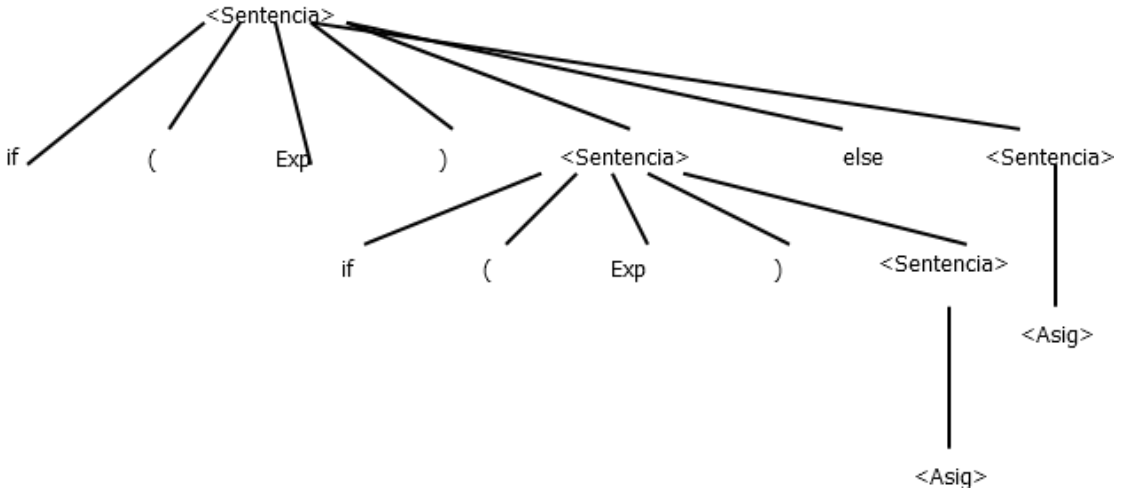
Análisis y Transformación de la gramática:

La Gramática de Minijava no es LL1 porque es:

- Es Ambigua.
- Es recursiva a izquierda.
- No está factorizada a izquierda.

Ejemplo donde la gramática es ambigua:





Eliminación de símbolos de BNF extendida:

```
1       <Inicial> ::= <Clase> <C> EOF
1.1     <C> ::= <Inicial> | e

2       <Clase>::= class idClase <H> {<Miembros>}
2.1     <H>::= <Herencia> | ε
2.2     <Miembros> ::= <Miembro> <Miembros> | ε

3       <Herencia>::= extends idClase

4       <Miembro>::= <Atributo> | <Ctor> | <Metodo>

 5      <Atributo>::= <Visibilidad> <Tipo> <ListaDecVars> ;

6       <Metodo>::= <FormaMetodo> <TipoMetodo> idMetVal <ArgsFormales> <Bloque>

7       <Ctor>::= idClase <ArgsFormales> <Bloque>

8       <ArgsFormales>::= ( <LAF> )
8.1     <LAF>::= <ListaArgsFormales> | ε

9       <ListaArgsFormales>::= <ArgFormal>

10      <ListaArgsFormales>::= <ArgFormal> , <ListaArgsFormales>

11      <ArgFormal>::= <Tipo> idMetVal

12      <FormaMetodo>::= static | dynamic

13      <Visibilidad>::= public | private

14      <TipoMetodo>::= <Tipo> | void

15      <Tipo>::= <TipoPrimitivo>  | <TipoReferencia>

16      <TipoPrimitivo>::= boolean  | char | int

17      <TipoReferencia>::= idClase | String | <TipoPrimitivo[]

18      <ListaDecVars>::= idMetVal

19      <ListaDecVars>::= idMetVal , <ListaDecVars>
```

```
20      <Bloque>::= { <S> }
20.1 <S>::= <Sentencia> <S> | ε

21      <Sentencia>::= ;
22      <Sentencia>::= <Asignacion> ;
23      <Sentencia>::= <SentenciaLlamada> ;
24      <Sentencia>::= <Tipo> <ListaDecVars> ;
25      <Sentencia>::= if ( <Expresion> ) <Sentencia>
26      <Sentencia>::= if ( <Expresion> ) <Sentencia>  else <Sentencia>
27      <Sentencia>::= while ( <Expresion> ) <Sentencia>
28      <Sentencia>::= <Bloque>
29      <Sentencia>::= return <Exp> ;
29.2    <Exp>::= <Expresion> | ε

30      <Asignacion>::= <AccesoVar> = <Expresion> | <AccesoThis> = <Expresion>

33      <Expresion>::= <ExpOr>

34      <ExpOr>::= <ExpOr> || <ExpAnd> | <ExpAnd>

35      <ExpAnd>::= <ExpIg>  && <ExpIg> | <ExpIg>

36      <ExpIg>::= <ExpComp> <OpIg> <ExpComp> | <ExpComp>

37      <ExpComp>::= <ExpAd> <OpComp> <ExpAd> |  <ExpAd>

38      <ExpAd>::= <ExpAd> <OpAd> <ExpMul> |  <ExpMul>

39      <ExpMul>::= <ExpUn> <RestoExpMul>
39.1    <RestoExpMul>::= <OpMul> <ExpUn> <RestoExpMul> |  ε

40      <ExpUn>::= <OpUn> <ExpUn> | <Operando>

41      <OpIg>::= == | !=

42      <OpComp>::= < | > | <= | >=

43      <OpAd>::= + | -

44      <OpUn>::= + | - | !

45      <OpMul>::= * | /

46      <Operando>::= <Literal> | <Primario>
```

```
48      <Literal>::= null | true | false | intLiteral | charLiteral | stringLiteral

49      <Primario>::= <ExpresionParentizada> | <AccesoThis> |
<LlamadaMetodoEstatico> | <LlamadaCtor> | <AccesoVar> | <LlamadaMetodo>

55      <ExpresionParentizada>::= ( <Expresion> ) <Enca>
55.1 <Enca>::= <Encadenado> |  ε

56      <AccesoThis>::= this <Enca>

57      <AccesoVar>::= idMetVal <Enca>

58      <LlamadaMetodo>::= idMetVal <ArgsActuales> <Enca>

59      <LlamadaMetodoEstatico>::= idClase . <LlamadaMetodo> <Enca>

60      <LlamadaCtor>::= new idClase <ArgsActuales> <Enca>

62       <LlamadaCtor>::= new <TipoPrimitivo> [ <Expresion> ] <Enca>

63      <ListaExps>::= <Expresion>

64      <ListaExps>::= <Expresion> , <ListaExps>

65      <Encadenado>::= . <LlamadaMetodoEstatico>

66      <Encadenado>::= . <AccesoVarEncadenado>

67      <Encadenado>::= <AccesoArregloEncadenado>

68      <AccesoVarEncadenado>::= idMetVal <Enca>

70      <AccesoArregloEncadenado>::= [ <Expresion> ] <Enca>
```

Observación: A partir de ahora pongo solo los elementos de la gramática que fueron modificados y no todos para hacer más sencillo el informe.

Eliminación de Recursión a Izquierda:

```
34      <ExpOr>::= <ExpAnd> <RestoExpOr>
34.1    <RestoExpOr>::= || <ExpAnd> <RestoExpOr> | ε

35      <ExpAnd>::= <ExpIg> <RestoExpAnd>
35.1    <RestoExpAnd>::= && <ExpIg> <RestoExpAnd> | ε

36      <ExpIg>::= <ExpComp> <RestoExpIg>
36.1    <RestoExpIg>::= <OpIg> <ExpComp> <RestoExpIg> | ε

37      <ExpComp>::= <ExpAd> <ExpCompFactorizada>
37.1     <ExpCompFactorizada>::= <OpComp> <ExpAd> | ε

38      <ExpAd>::= <ExpMul> <RestoExpAd>
38.1    <RestoExpAd>::= <OpAd> <ExpMul> <RestoExpAd> | ε

39      <ExpMul>::= <ExpUn> <RestoExpMul>
39.1    <RestoExpMul>::= <OpMul> <ExpUn> <RestoExpMul> | ε
```

Factorización:

Observación: Acá si inclui toda la gramática de minijava porque es más sencillo de leer en el último caso como quedó finalmente la misma.

```
1       <Inicial> ::= <Clase> <C> EOF
1.1     <C> ::= <Inicial> | ε

2       <Clase>::= class idClase <H> {<Miembros>}
2.1     <H>::= <Herencia> | ε
2.2      <Miembros> ::= <Miembro> <Miembros> | ε

3       <Herencia>::= extends idClase

4       <Miembro>::= <Atributo> | <Ctor> | <Metodo>

5       <Atributo>::= <Visibilidad> <Tipo> <ListaDecVars> ;

6       <Metodo>::= <FormaMetodo> <TipoMetodo> idMetVal <ArgsFormales> <Bloque>

7       <Ctor>::= idClase <ArgsFormales> <Bloque>

8       <ArgsFormales>::= ( <LAF> )
8.1     <LAF>::= <ListaArgsFormales> | ε

9       <ListaArgsFormales>::= <ArgFormal> <LAFS>
9.1     <LAFS>::= , <ListaArgsFormales> | ε

11      <ArgFormal>::= <Tipo> idMetVal

12      <FormaMetodo>::= static | dynamic

13      <Visibilidad>::= public | private

14      <TipoMetodo>::= <Tipo> | void

15      <Tipo>::= boolean <Bool> | char <Ch> | int <I> | idClase | String
15.1    <Bool>::= [] | ε
15.2    <Ch>::= [] | ε
15.3    <I>::= [] | ε

16      <TipoPrimitivo>::= boolean | char | int

17      <TipoReferencia>::= idClase | String | <TipoPrimitivo>

18      <ListaDecVars>::= idMetVal <LDVFact>
18.1    <LDVFact>::= , <ListaDecVars> | ε
```

```
20      <Bloque>::= { <S> }
20.1 <S>::= <Sentencia> <S> | ε


21      <Sentencia>::= ; | <Asignacion> ;| <SentenciaLlamada> ;| <Tipo> <ListaDecVars>
;| if ( <Expresion> ) <Sentencia> <SentenciaFact> | while ( <Expresion> ) <Sentencia> |
<Bloque> | return <Exp> ;
21.1    <SentenciaFact>::= else <Sentencia> | ε
21.2    <Exp>::= <Expresion> | ε

30      <Asignacion>::= <AccesoVar> = <Expresion> | <AccesoThis> = <Expresion>

32      <SentenciaLlamada>::= ( <Primario> )

33      <Expresion>::= <ExpOr>

34      <ExpOr>::= <ExpAnd> <RestoExpOr>
34.1    <RestoExpOr>::= || <ExpAnd> <RestoExpOr> | ε

35      <ExpAnd>::= <ExpIg> <RestoExpAnd>
35.1    <RestoExpAnd>::= && <ExpIg> <RestoExpAnd> | ε

36      <ExpIg>::= <ExpComp> <RestoExpIg>
36.1    <RestoExpIg>::= <OpIg> <ExpComp> <RestoExpIg> | ε

37      <ExpComp>::= <ExpAd> <ExpCompFactorizada>
37.1 <ExpCompFactorizada>::= <OpComp> <ExpAd> | ε

38      <ExpAd>::= <ExpMul> <RestoExpAd>
38.1    <RestoExpAd>::= <OpAd> <ExpMul> <RestoExpAd> | ε

39      <ExpMul>::= <ExpUn> <RestoExpMul>
39.1    <RestoExpMul>::= <OpMul> <ExpUn> <RestoExpMul> | ε

40      <ExpUn>::= <OpUn> <ExpUn> | <Operando>

41      <OpIg>::= == | !=

42      <OpComp>::= < | > | <= | >=

43      <OpAd>::= + | -

44      <OpUn>::= + | - | !

45      <OpMul>::= * | /

46      <Operando>::= <Literal> | <Primario>

48      <Literal>::= null | true | false | intLiteral | charLiteral | stringLiteral
```

```
49      <Primario>::= <ExpresionParentizada> | <AccesoThis> |
<LlamadaMetodoEstatico> | <LlamadaCtor> | idMetVal <PrimarioFactorizado>
49.1    <PrimarioFactorizado>::= <ArgsActuales> <Enca> | <Enca>

55      <ExpresionParentizada>::= ( <Expresion> ) <Enca>
55.1 <Enca>::= <Encadenado> | ε

56      <AccesoThis>::= this <Enca>

57      <AccesoVar>::= idMetVal <Enca>

58      <LlamadaMetodo>::= idMetVal <ArgsActuales> <Enca>

59      <LlamadaMetodoEstatico>::= idClase . <LlamadaMetodo> <Enca>

60      <LlamadaCtor>::= new <LlamadaCtorFactorizado>
60.1    <LlamadaCtorFactorizado>::= idClase <ArgsActuales> <Enca> | <TipoPrimitivo> [
<Expresion> ] <Enca>

62      <ArgsActuales>::= ( <LE> )
62.1    <LE>::= <ListaExps> | ε

63      <ListaExps>::= <Expresion> <ListaExpsFact>
63.1    <ListaExpsFact>::= , <Expresion> | ε

65      <Encadenado>::= . idMetVal <EncadenadoFactorizado> | [ <Expresion> ] <Enca>
65.1    <EncadenadoFactorizado>::= <ArgsActuales> <Enca> | <Enca>

68      <LlamadaMetodoEncadenado>::= idMetVal <ArgsActuales> <Enca>

69      <AccesoVarEnca>::= idMetVal <Enca>

70      <AccesoArregloEncadenado>::= [ <Expresion> ] <Enca>
```

Estructura del analizador sintáctico:

- Analizador Sintáctico:
  - Recibe un archivo.
  - Se encarga de iniciar el analizador.
  - Le pasa el archivo.
  - Le pide al analizador tokens.
  - Analiza el token actual dependiendo de la gramática.
- Analizador Léxico:
  - Abre el archivo.
  - Lee el archivo una sola vez y devuelve un token o un error en caso de encontrar uno.
  - utiliza el método getToken() para cumplir la función anterior.
- Token:
  - Es la clase que estructura al token.
  - Contiene el nombre, lexema, el número de columna y el número de fila.
- Excepciones:
  - ExAndMalFormado.
  - ExCaracterInvalido.
  - ExCaracterMalFormado.
  - ExComentarioMalFormado.
  - ExNumeroMalFormado.
  - ExOrMalFormado.
  - ExStringMalFormado.

Esquema de Traducción:

El siguiente paso fue la construcción del esquema de traducción, construido apatrir de la gramática resultante del paso anterior. El esquema cuenta con las acciones semánticas destinada a la construcción de la tabla de símbolos y los AST necesarios para los chequeos semánticos y la generación de código en si. Las acciones semánticas se indican entre llaves.

```
1       <Inicial> -> <Clase> <C> EOF
1.1     <C> -> <Inicial>
1.2     <C> -> e

2       <Clase>-> class idClase <H> {<Miembros>}

2.1     <H> -> <Herencia>
2.1.1   <H> -> e

2.2     <Miembros> -> <Miembro> <Miembros>
2.2     <Miembros> -> e

3       <Herencia>-> extends idClase

4.1     <Miembro> -> <Atributo>
4.2     <Miembro> ->  <Ctor>
4.3     <Miembro> -> <Metodo>

5       <Atributo> -> <Visibilidad> <Tipo> <ListaDecVars> ;
                      {cada elemento L de la ListaDevVars lo agrego como variable de la
                      clase actual junto con el tipo}

6       <Metodo> -> <FormaMetodo> <TipoMetodo> idMetVal <ArgsFormales> <Bloque>
                      {primero creó el Método M, se lo seteo a la clase actual y luego de
                crear el Bloque B se lo paso como cuerpo al método}

7       <Ctor> -> idClase <ArgsFormales>
                {Seteo el constructor por defecto como falso, y seteo el actual a la clase}
                    <Bloque>
                {Seteo el bloque B como cuerpo de la unidad actual.}

8       <ArgsFormales> -> ( <LAF> )
8.1     <LAF> -> <ListaArgsFormales>
8.1.1   <LAF> -> e

9       <ListaArgsFormales> -> <ArgFormal> <LAFS>
9.1     <LAFS> -> , <ListaArgsFormales>
```

```
9.1.1   <LAFS> -> e


11      <ArgFormal> -> <Tipo> idMetVal

12.1    <FormaMetodo> -> static
12.2    <FormaMetodo> -> dynamic
13      <Visibilidad>-> public | private

14.1    <TipoMetodo>-> <Tipo>
14.2    <TipoMetodo> -> void

15      <Tipo>-> boolean <Bool>
15      <Tipo>-> char <Ch>
15      <Tipo>-> int <I>
15      <Tipo>-> idClase
15      <Tipo>-> String
15.1    <Bool>-> [] | e
15.2    <Ch>-> [] | e
15.3    <I>-> [] | e

18      <ListaDecVars>-> idMetVal <LDVFact>
18.1    <LDVFact>-> , <ListaDecVars> | e

20      <Bloque> ->
                        {
                        {-Guardo el bloque previo como Previo
                         -Creo el bloque nuevo como actual.
                         - Seteo el bloque actual de la ts a actual.
                         }
                         <S> }
                        {Vuelvo a setear el bloque Previo como bloque actual de la ts.}

20.1 <S> -> <Sentencia>
                {Se setea la nueva sentencia al bloque actual.}
              <S> | e


21      <Sentencia>-> ;
                        {se retorna una SentenciaVacia}
21      <Sentencia>->  <Asignacion>
                        {Se crea un nodo asignacion}
                        ;
                        {Se retorna el nodo asignacion anterior}
21      <Sentencia>->  <SentenciaLlamada>
```

```
                    {Se crea el nodo SentenciaLlamada}
                    ;
                    {Se retorna el anterior nodo SentenciaLlamada}
21      <Sentencia>->  <Tipo>
                    {Se busca el Tipo t}
                    <ListaDecVars>
                    {Se agregan en una lista las variables}
                    ;
                    {Se agregan todas las variables en la lista al bloque actual}
                    {Se retorna un DecVars con la lista de variables y el tipo t}
21      <Sentencia>-> if (
                     {Se crea un nuevo NodoIf con el token "if"}
                     <Expresion>
                     ) <Sentencia>
                    <SentenciaFact>
                    {Se retorna un nodo SentenciaFactorizada con la sentencia, la
            expresion y el token “if”}

21      <Sentencia>->  while ( <Expresion> ) <Sentencia>
                    {Se retorna un nodo While con la sentencia, la expresion y el token
                    “while”}
21      <Sentencia>-> <Bloque>
                    {Se retorna un nodo Bloque con el token en la “{”}

21      <Sentencia>->  return <Exp> ;
                    {Se retorna un nodo Return con el token “return” y la expresion}


21.1    <SentenciaFact>-> else <Sentencia>
                    {Se retorna un nodo Else con el token de “if” pasado por parametro,
                    la sentencia del if y la expresion tambien del if, ademas tambien se
                    guarda la sentencia del else}
21.1    <SentenciaFact>-> e
                    {Se retorna un nodo “IF” con el token “if”, la expresion y la sentencia
                    pasados por parametro}


21.2    <Exp> -> <Expresion>
21.2    <Exp> -> e

30      <Asignacion> ->
                    {Se setea en la TS que se esta procesando un lado izq}
                    <AccesoVar>
                    {Se crea un Nodo AccesoVar}
                    =
```

```
                    {Se setea que no se procesa un lado izq en la TS}
                    <Expresion>
                    {Se crea un nodo Expresion}
                    {Se retorna un nodo Asignacion con el token en el nodo acceso, la
                    expresion}
30      <Asignacion> ->
                    {Se setea en la TS que se está procesando un lado izq}
                    <AccesoThis>
                    =
                    {Se setea en la TS que no se está procesando un lado izq}
                     <Expresion>
                    {Se retorna un nodo Asignacion con el token en el nodo acceso, la
                    expresion}

32      <SentenciaLlamada>-> ( <Primario> )
                    {Se retorna una SentSimple con el token al primer parentesis y la
                    expresion retornada por Primario}

33      <Expresion>-> <ExpOr> {Se retorna el NodoExpresion que retorna ExpOr}

34      <ExpOr>-> <ExpAnd> {Se guarda el NodoExpresion que retorna}
                    {Se llama a RestoExpOr con el nodo guardado}
                    <RestoExpOr>
                    Se retorna el NodoExpresion que retorna RestoExpOr}

34.1    <RestoExpOr> -> ||  {Se crea un nuevo ExpAnd con el token}
                            <ExpAnd>
                            {Se crea un nuevo NodoExpBinaria con el ExpAnd y el
                            NodoExpresion recibido como parametro como lado
                            izquierdo}
                             <RestoExpOr>
                            {Se retorna el NodoExpresion que retorna RestoExpOr}
34.1    <RestoExpOr> -> e {Se retorna el NodoExpresion recibido como parametro}

35      <ExpAnd> -> <ExpIg>  {Se crea un nuevo ExpIg con el token}
                    {Se llama a RestoExpAnd con el nodo guardado}
                    <RestoExpAnd>
                    {Se retorna el NodoExpresion que retorna ExpIg}


35.1    <RestoExpAnd>-> && {Se crea un nuevo ExIg con el token}
                        <ExpIg> {Se crea un nuevo NodoExpBinaria con el token, la
                    expresion pasada por param y la ExpIg}
                            <RestoExpAnd>
                            {Se retorna el nodo Expresion}
```

```
35.1    <RestoExpAnd> -> e {Se retorna el nodo pasado por parametro}

36      <ExpIg> -> <ExpComp> {Se crea NodoExp}
                        {Se llama al met restoExpIg con el NodoExp}
                  <RestoExpIg> {Se retorna el nuevo NodoExp de la llamada anterior}

36.1    <RestoExpIg> -> <OpIg> <ExpComp> {Se crea un NodoExp e2}
                        {Se crea un NodoExpBinaria con el token, la exp e pasada por
                  parametro y el nodo e2}
                        <RestoExpIg>  {Se retorna el nodo creado por la llamada
                  restoExpIg y el nodo NodoExpBinaria pasada por parametro}
36.1    <RestoExpIg> -> e {Se retorna el nodo e pasado por parametro}

37      <ExpComp>-> <ExpAd> <ExpCompFactorizada>
37.1 <ExpCompFactorizada>-> <OpComp> <ExpAd>
37.1 <ExpCompFactorizada> -> e

38      <ExpAd> -> <ExpMul> {Se crea un NodoExp e con la llamada a ExpMul }
.                       <RestoExpAd> {Se llama a restoExpMul con el nodo e y luego se .
                        retorna el NodoExp generado por la llamada}

38.1    <RestoExpAd> ->
                        <OpAd>
                        <ExpMul>  {Se crea un NodoExp e2}
                        {Se crea un NodoExpBinaria con el nodo e2, el nodoExp pasado
                        por parametro y el token}
                        <RestoExpAd> {Se llama a restoExpAd con el NodoExpBinaria y se
                        retorna luego de la llamada}
38.1    <RestoExpAd> -> e {Se retorna el nodoExp e pasado por parametro}

39      <ExpMul>->
                        {Se crea un NodoExp e con la llamada a expUn }
                        <ExpUn>
                        {Se llama a restoExpMul con el nodo e}
                        <RestoExpMul>
                        {Se retorna el nodo generado por la llamada restoExpMul}
39.1    <RestoExpMul>->
                        {Se llama al metodo opMul}
                        <OpMul>
                        {Se crea un NodoExp e2 con la llamada a expUn}
                        <ExpUn>
                        {Se crea un NodoExpBinara expBinaria con el token, el NodoExp e
                        pasado por param y el nodo e2}
                        <RestoExpMul>
                        {Se retorna el nodo generado por la llamada restoExpMul }
```

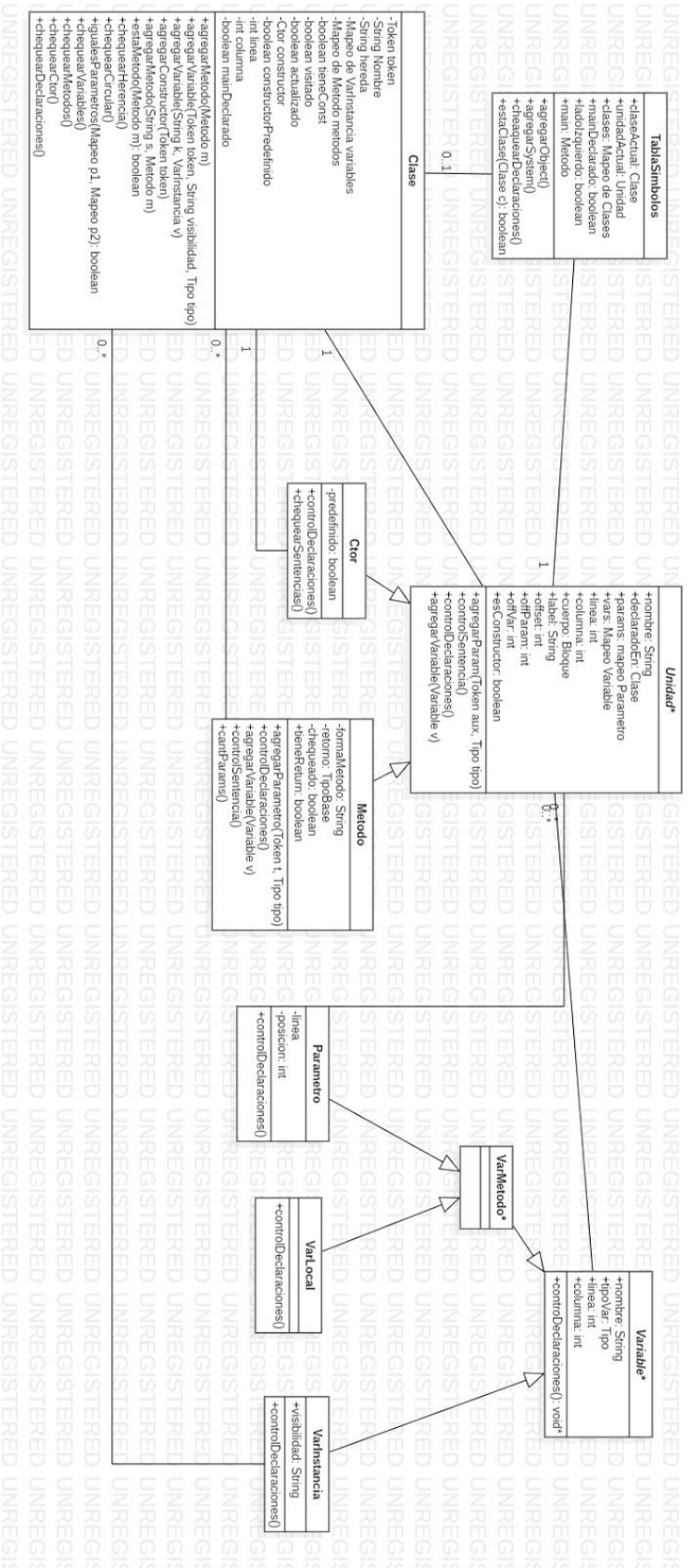
```
39.1    <RestoExpMul> -> e {Se retorna el nodoExp e pasado por parametro}

40      <ExpUn>-> <OpUn>
                        {Se crea un nodoExp e de la llamada a expUn}
                        <ExpUn>
                        {Se retrona un NodoExpUnaria con el token y e}

40      <ExpUn> ->
                        {Se llama a operando que retorna un NodoExp}
                        <Operando>
                        {Se retorna el nodo generado por la llamada}

41      <OpIg> -> ==
41      <OpIg> -> !=

42      <OpComp>-> <
42      <OpComp>-> >
42      <OpComp>-> <=
42      <OpComp>-> >=

43      <OpAd>-> +
43      <OpAd>-> -

44      <OpUn>-> +
44      <OpUn>-> -
44      <OpUn>-> !

45      <OpMul>-> *
45      <OpMul>-> /

46      <Operando>-> <Literal>
46      <Operando>-> <Primario>

48      <Literal> -> null {Retorna un NodoLiteral con el token}
48      <Literal> -> true {Retorna un NodoLiteral con el token}
48      <Literal> -> false {Retorna un NodoLiteral con el token}
48      <Literal> -> intLiteral {Retorna un NodoLiteral con el token}
48      <Literal> -> charLiteral {Retorna un NodoLiteral con el token}
48      <Literal> -> stringLiteral {Retorna un NodoLiteral con el token}

49      <Primario> -> <ExpresionParentizada>
                        {Retorna un NodoExp de la llamada a   expresionParentizada}
49      <Primario> -> <AccesoThis>
                        {Retorna un NodoExp de la llamada accesoThis}
49      <Primario> ->  <LlamadaMetodoEstatico>
```

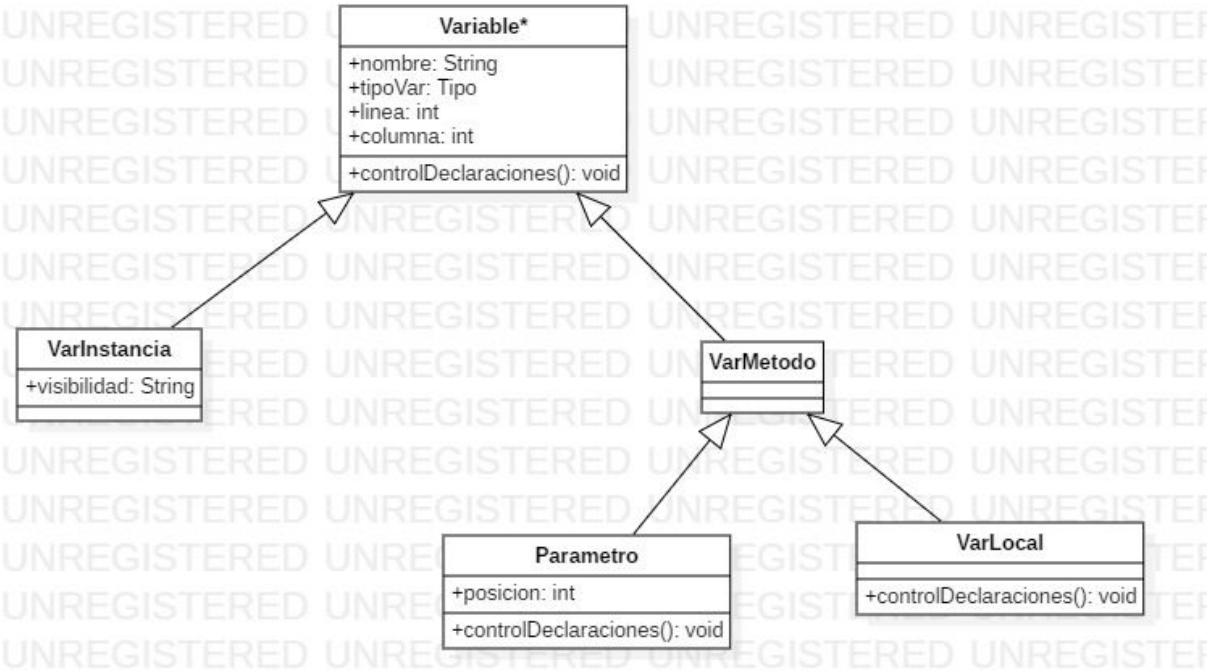
```
                        {Retorna un NodoExp de la llamada}
49      <Primario> -> <LlamadaCtor>
                        {Retorna un NodoExp de la llamada}
49      <Primario> -> idMetVal <PrimarioFactorizado>
                        {Retorna un NodoExp de la llamada}


49.1    <PrimarioFactorizado> -> <ArgsActuales> <Enca>
                                 {Retorna un NodoPrimario de con el id pasado por
                                 parametro, Argumentos, y el encadenado}
49.1    <PrimarioFactorizado> -> <Enca>
                                 {Retorna un NodoPrimario de la llamada a
                                 NodoLlamadaDirecta}

55      <ExpresionParentizada>-> ( <Expresion> ) <Enca>
                                 {Retorna un NodoExp con el token, la Expresion y el
                                 encadenado de las llamadas pertinentes}
55.1 <Enca>-> <Encadenado> {Retrona el Encadenado de la llamada encadenado }

56      <AccesoThis>-> this <Enca>

57      <AccesoVar>-> idMetVal <Enca>

58      <LlamadaMetodo>-> idMetVal <ArgsActuales>
                                 {Retorna Arguementos de la llamada argsActuales}

59      <LlamadaMetodoEstatico>-> idClase .
                                 {Crea Argumentos a llamando llamadaMetodo}
                                         <LlamadaMetodo>
                                 {Crea Encadenado con la llamada enca}
                                          <Enca>
                                 {Crea NodoLlamadaConClase con el idClase, y a}
                                 {Si el encadenado no es nulo, los agrega al
                                 NodollamadaConClase}
                                 {Retorna el nodoLlamadaConClase}
60      <LlamadaCtor>-> new <LlamadaCtorFactorizado>
                        {Retorna NodoPrimario de la llamadaCtorFactorizado}

60.1    <LlamadaCtorFactorizado> -> idClase
                                        {Crea Argumentos a}
                                        <ArgsActuales>
                                        {Crea Encadenado enca}
                                        <Enca>
                                        {Crea NodoCtorComun con id, a}
                                        {si el enca no es nulo lo setea al nodo}
```

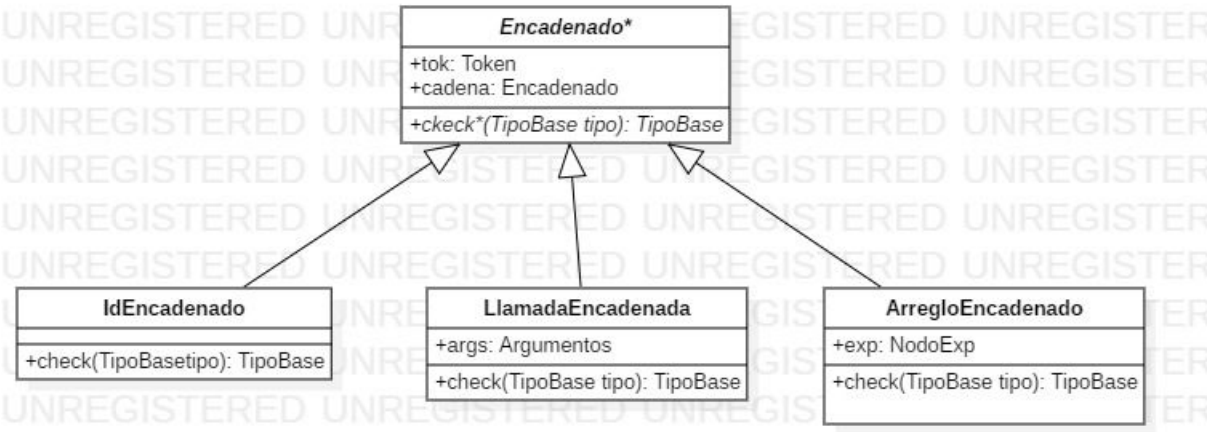
```
                                        {retorna el nodo}
60.1    <LlamadaCtorFactorizado> -> <TipoPrimitivo>
                                        {Crea Tipo t de la llamada tipoPrimitivo}
                                        [ <Expresion>
                                        {Crea un NodoExp de la llamada expresion}]
                                        <Enca>
                                        {Crea un nodo Encadenado de la llamada enca}
                                        {Crea un NodoCtorArreglo con el id , exp}
                                        {Setea el encadenado}
                                        {Retorna el NodoCtorArreglo}



62      <ArgsActuales> -> (
                        {Crea Argumentos con la llamada a LE}
                        <LE> )
                        {Retorna Arguementos}
62.1    <LE> -> <ListaExps>
                {Retorna Argumentos de la llamada listaExps}
62.1    <LE> -> e
                {Retorna un nuevo Argumentos}

63      <ListaExps>->
                        {Crea un NodoExp con la expresion}
                        <Expresion> <ListaExpsFact>
                        {Retorna Argumentos con la llamada listaExpsFact y la expresion
                        pasada por parametro}
63.1    <ListaExpsFact>-> , {Crea Argumentos ar con la llamada expresion}
                              <Expresion>
                                 {Retorna una nueva Argumentos con ar y la expresion
                        pasada por parametro}
63.1    <ListaExpsFact>-> e
                        {retorna un nuevo Argumentos con la expresion pasada por
                        parametro y null}

65      <Encadenado>-> . idMetVal <EncadenadoFactorizado>
                        {Retorna Encadenado de la llamada encadenadoFactorizado con el
                        id de idMetVal pasado por parametro}
65      <Encadenado>->  [ <Expresion> ] <Enca>
                        {Retorna un Encadenado de la llamada ArregloEncadenado con el
                        id, el encadenado y la expresion de las llamadas}
65.1    <EncadenadoFactorizado>-> <ArgsActuales> <Enca>
                        {Crea un LlamadaEncadenada con el id, Argumentos y el
                        encadenado de las llamadas y luego lo retrona}
65.1    <EncadenadoFactorizado>->  <Enca>
```

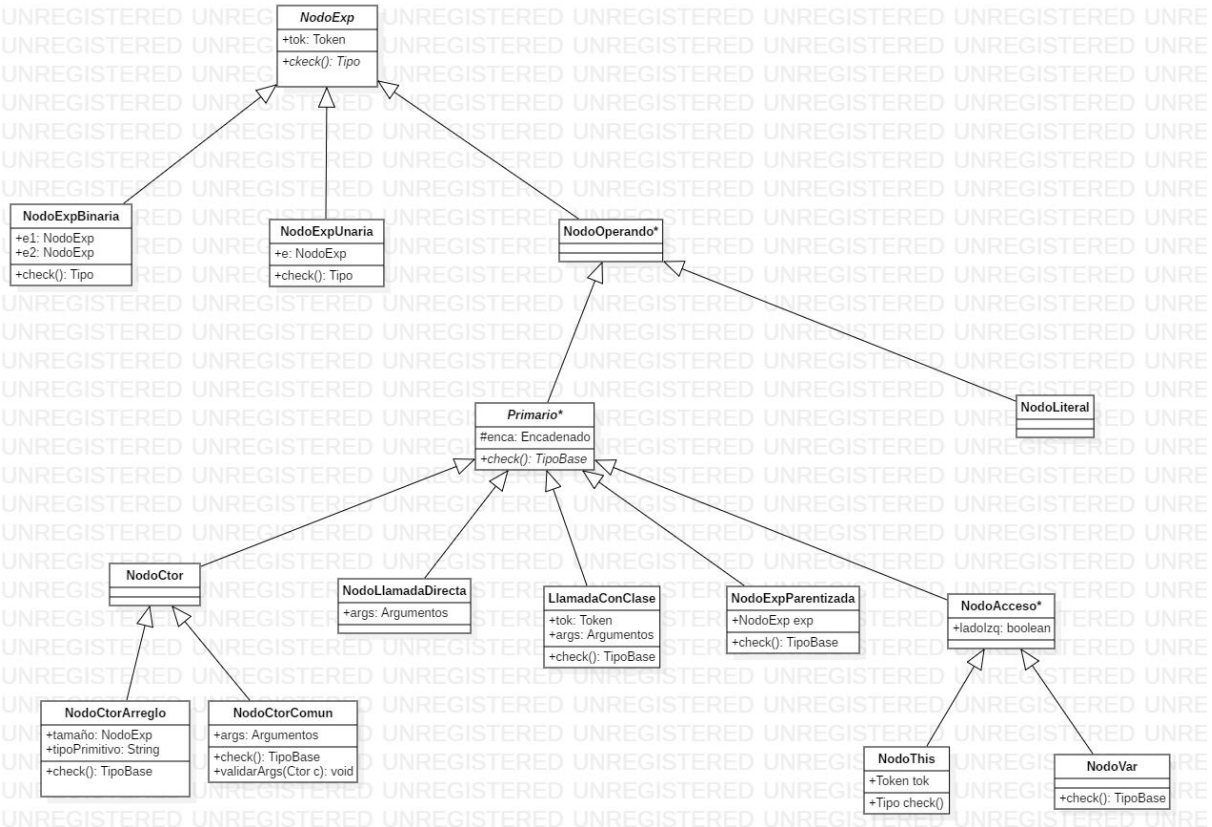
{Retorna el Encadenado de la llamada idEncadenado con el id y el encadenado de la llamadas}

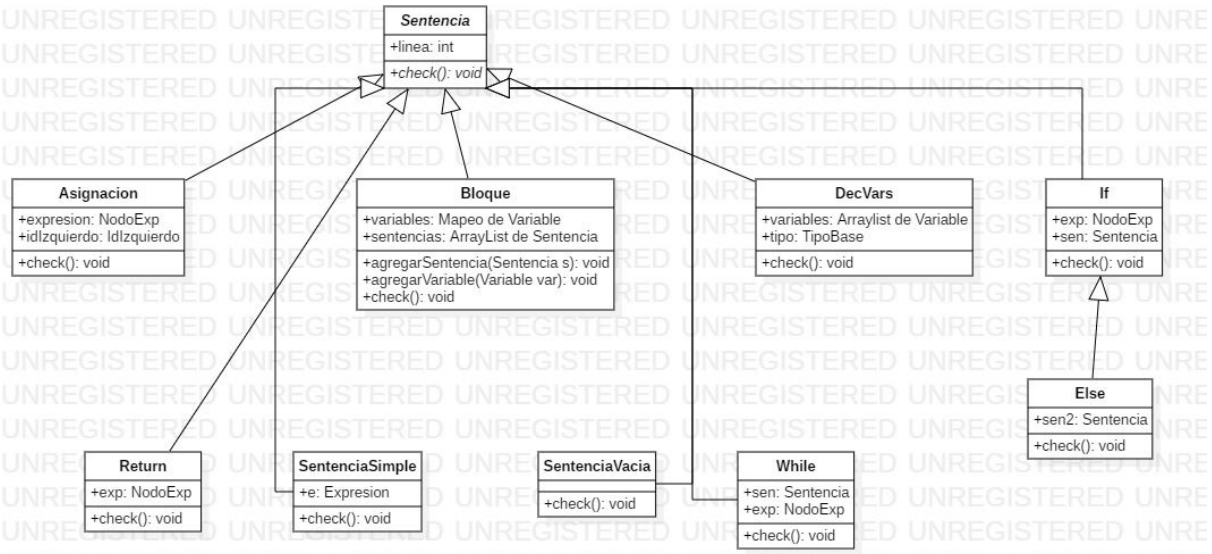
Diagrama de clases:




Variable*
+nombre: String
+tipoVar: Tipo
+linea: int
+columna: int
+controlDeclaraciones(): void
VarInstancia
+visibilidad: String
VarMetodo
Parametro
+posicion: int
+controlDeclaraciones(): void
VarLocal
+controlDeclaraciones(): void


Encadenado*
+tok: Token
+cadena: Encadenado
+ckeck*(TipoBase tipo): TipoBase
IdEncadenado
+check(TipoBasetipo): TipoBase
LlamadaEncadenada
+args: Argumentos
+check(TipoBase tipo): TipoBase
ArregloEncadenado
+exp: NodoExp
+check(TipoBase tipo): TipoBase


NodoExp
+tok: Token
+ckeck(): Tipo
NodoExpBinaria
+e1: NodoExp
+e2: NodoExp
+check(): Tipo
NodoExpUnaria
+e: NodoExp
+check(): Tipo
NodoOperando*
NodoLiteral
Primario*
#enca: Encadenado
+check(): TipoBase
NodoCtor
NodoLlamadaDirecta
+args: Argumentos
LlamadaConClase
+tok: Token
+args: Argumentos
+check(): TipoBase
NodoExpParentizada
+NodoExp exp
+check(): TipoBase
NodoAcceso*
+ladoIzq: boolean
NodoCtorArreglo
+tamaño: NodoExp
+tipoPrimitivo: String
+check(): TipoBase
NodoCtorComun
+args: Argumentos
+check(): TipoBase
+validarArgs(Ctor c): void
NodoThis
+Token tok
+Tipo check()
NodoVar
+check(): TipoBase


Sentencia
+linea: int
+check(): void
Asignacion
+expresion: NodoExp
+idIzquierdo: IdIzquierdo
+check(): void
Bloque
+variables: Mapeo de Variable
+sentencias: ArrayList de Sentencia
+agregarSentencia(Sentencia s): void
+agregarVariable(Variable var): void
+check(): void
DecVars
+variables: Arraylist de Variable
+tipo: TipoBase
+check(): void
If
+exp: NodoExp
+sen: Sentencia
+check(): void
Else
+sen2: Sentencia
+check(): void
Return
+exp: NodoExp
+check(): void
SentenciaSimple
+e: Expresion
+check(): void
SentenciaVacia
+check(): void
While
+sen: Sentencia
+exp: NodoExp
+check(): void

Explicación de los Métodos de la Tabla de Símbolos:

Por una cuestión de practicidad explico los métodos que no son getters y setters, porque su definición es de público conocimiento.

- agregarObject(): Crea una clase con el nombre Object. Donde eventualmente todas las clases heredarán de esta clase.
- agregarSystem(): Crea una clase con el nombre System y sus respectivos métodos.
- estaClase(String clase): Devuelve verdadero o falso dependiendo si la tabla contiene al nombre de la clase pasada por parámetro.
- chequearDeclaraciones(): Para cada clase “c” que contenga la tabla, si c es distinto de la clase Object o System, le pide que cheque sus declaraciones.
  Luego de que cada clase haya chequeado sus declaraciones, se fija si alguna de estas tenía un método *main* declarado, de no ser así, lanza una excepción.
- chequearSentencias(): Inicia el generador de código, indicando donde se encuentra el método main. Y luego para cada clase llama al metodo chequearSentencias() que será explicado en breve.

Explicación de los chequeos que realiza el compilador

En la primera pasada, el compilador, a medida que se realiza el análisis sintáctico del archivo de entrada, procesa todas las declaraciones y construye una tabla de símbolos. A su vez, cada método, creó su correspondiente ATS.

Chequeo de declaraciones de una clase:

La tabla de símbolos realiza la llamada al chequeo de declaraciones de cada una de sus clases en el método chequearDeclaraciones().

se verifica que:

1. Chequear que no tenga 2 métodos con el mismo nombre.
2. Si la clase define un método con el mismo nombre que un método de alguna clase ancestra, entonces deberá chequear que el nuevo método tenga:
    a. Misma cantidad y tipo de parámetros.
    b. Mismo tipo de retorno.
    c. Misma forma.
3. Chequear que ningun metodo tenga el mismo nombre que una variable de instancia
4. chequear que no haya herencia circular
5. chequear las declaración de cada uno de los métodos.

Chequeo de declaraciones de un método:

El chequeo de declaraciones de un método consiste en:

1. Chequear que el tipo de retorno sea válido.
2. Chequear las declaraciones de cada uno de sus parámetros.

Chequeo de declaraciones de un parámetro:

El chequeo de declaraciones de un parámetro consiste en:

1. Chequear que su tipo sea válido.

Chequeo en sentencias:

- If: Consiste en chequear que el tipo de la expresión sea boolean y chequear la sentencia.
- Else: Consiste en chequear que el tipo de la expresión sea boolean y chequear ambas sentencias (la del if y la del else).
- Sentencia simple: Consiste en chequear la expresión
- Return: Consiste en chequear:
  - Que no se esté intentando retornar un valor dentro de un constructor.
  - Que no se esté intentando retornar un valor dentro un método void.
  - Si la expresión no es nula, se chequea la expresión.
- DecVars: Se chequea cada una de las variables.
- Asignación: Se chequea que el tipo de la expresión del lado izquierdo de la asignación se compatible con el tipo de la expresión del lado derecho.
- While: Se chequea que la expresión sea de tipo boolean y se chequea la sentencia.

Chequeo en expresiones:

- ExpBinaria: A partir del operador de la expresión, se chequea que este sea aplicable a partir de los tipos de las 2 expresiones adyacentes a este.
- ExpUnario: Se chequea que el tipo de la expresión sea boolean si el operador es '!' o que sea de tipo int si el operador es '+' o '-'.
- This: Se chequea que no se esté intentando utilizar dentro de un método estático. El chequeo de este nodo devuelve un objeto de tipo TipoClase con la clase actual.
- Literal: Devuelve un objeto del tipo que corresponda según el literal.
- Construir: Chequea que la tipo del objeto que se está queriendo crear exista en la tabla de símbolos y se verifican que los argumentos correspondan con los del constructor.
- LlamadaVar: Se controla que la variable esté entre las variables locales del método actual o que esté dentro de las variables de instancia de la clase.
- ExpConEncadenado: Se chequea la expresión.
- LlamadaConClase: Se chequea la clase a la que se referencia exista en la tabla de símbolos, y de existir, que posea el método estático que se está queriendo usar.

Cálculo de offsets y etiquetas

Generacion de las etiquetas de las unidades:

Las etiquetas de los métodos son generadas por demanda. Es decir, cuando se solicita la etiqueta de un método, si esta no fue generada anteriormente, se genera y se retorna, en caso contrario solo se retorna la etiqueta ya generada.

Calculo de Offset de las variables de instancia:

El offset de las variables de instancia se calcula a partir del offset de las variables de instancia de la clase ancestra. Por ejemplo, supongamos 2 clases A y B, donde B hereda de A y ambas tienen 3 variables de instancia, los offset de las variables de instancia A iran de 0 a 2, mientras que los de B estarán en el rango de 3 a 5.

Calculo de los offsets de parámetros:

El offset de los parámetros se calcula de la siguiente manera:
si el parámetro pertenece a un metodo estatico, entonces
Param.offset = 2 + cantidadParametrosDeMetodo - posicionDelParametro.
sino (el método es dinámico)
Param.offset = 3 + cantidadParametrosDeMetodo - posicionDelParametro

Calculo de los offset de variables locales:

Los offsets de las variables locales son seteados al realizar el chequeo de declaraciones de las mismas.

Calculo de los offsets de unidades:

Los offsets de las unidades son calculados al realizar el chequeo de herencia circular. de modo que al encontrarse con un metodo, si este está siendo redefinido , se le pone le mismo offset que el que tenía en la clase ancestra. En caso de que un método heredado no sea redefinido, este preservara el mismo offset.

Para los métodos propios de una clase (ni heredados, ni redefinidos), el offset se calcula a partir de la cantidad de métodos dinámicos de la clase ancestra.

Generación de Código:

Clases donde se genera el código:
Cada clase es la responsable de generar el código que corresponde a la vez que realiza su chequeo de sentencias.

- ArregloEncadenado
- Bloque
- BloqueAux
- Bool
- Cadena
- Char
- Clase
- Ctor
- DecVars
- Else
- IdEncadenado
- IdIzqEncadenado
- If
- Int
- LlamadaEncadenada
- Metodo
- NodoCtorArreglo
- NodoCtorComun
- NodoExpBinaria
- NodoExpUnaria
- NodoLlamadaConClase
- NodoLlamadaDirecta
- NodoLlamadaVar
- NodoVar
- NodoThis
- Return
- SentSimple
- TipoString
- While

Casos de Test: Los casos de test se separan en dos partes, los correctos y los que se detectan errores:
Casos Correctos:

- 01: Se testean los tipos primitivos y las operaciones para ellas.
- 02: Se testean los tipos arreglos (boolean, char e int), se los crea, inicializa, insertan valores y por último se muestran en consola.
- 03: Se testean distintas clases y como una puede usar elementos de la otra.

- 04: Se referencia a un método redefinido.
- 05: Utilización de un while.
- 06: Utilización de un if.
- 07: Utilización de variables heredadas.
- 08: Referencia a This.
- 09: LLamada a métodos estáticos.
- 10: Llamada a un método de un objeto con un constructor predefinido.
- 11: Llamada a un método sin retorno.
- 12: Llamada a un constructor con System.read().