Proyecto Compiladores e Interpretes

Etapa 4

Analizador Semántico

Rodrigo Díaz Figueroa

LU 97400

Clases utilizadas y decisiones de diseño

Cambios desde la última etapa:

Se agregaron las clases correspondientes al AST y Operadores y Tipos.

Nuevas clases:

Correspondientes al AST:

***Sentencias*:**

NodoSentencia
NodoBloque
NodoDeclaracionVars
NodoIf
NodoWhile
NodoSentenciaSimple
NodoPuntoComa
NodoAsignacion
NodoReturn

Expresiones:

NodoExpresion
NodoExpBin
NodoExpUn
NodoOperando
NodoPrimario
NodoLiteral
NodoLitEnt
NodoLitNull
NodoLitBool
NodoLitChar
NodoLitString
NodoLlamDirect
NodoLlamEstat
NodoExpParent
NodoConst
NodoConstArray
NodoConstComun
NodoAcceso
NodoVar
NodoThis

Encadenados:

Encadenado
NodoVarEncad
NodoLlamEncad
NodoArregloEncad

Operadores:

Operador
OpMat
OpBool
OpComp
OpCompMat

Tipos:

Tipo
TipoVoid
TipoReferencia
TipoInt
TipoBool
TipoChar
TipoString
TipoClase
TipoNull

Decisiones de diseño:

- En la inicializacion de un atributo de clase, no es posible utilizar otros atributos o metodos dinamicos de la misma clase, ni la palabra reservada “this” para hacer referencia a la propia clase, es decir, los atributos solo pueden ser inicializados con expresiones, constantes, constructores, metodos estaticos de la misma clase o metodos accedidos a través de un objeto creado.

- La Tabla de Símbolos mantiene una referencia a la clase actual, miembro actual y al bloque actual, que son utilizadas durante los chequeos se sentencias.

- Se optó por utilizar solo un tipo de excepción para los errores semánticos y utilizar el mensaje para especificar el tipo de error.

Logros

Los logros que se intentan alcanzar de esta etapa son:

- **Imbatibilidad Semántica II.**
- **Arreglo de todo bien usado.**
- **Inicializaciones inline controladas.**
- **Llamadas sobrecargadas.**
- **Todos los caminos llevan al retorno.**

Compilación y Ejecución

Para compilar el programa desde consola se deberá ejecutar el siguiente comando

```
>javac Minijavac.java
```

Observación: Si este comando no compila todas las clases del programa se deberá ejecutar entonces

```
>javac Minijavac.java AnalizadorSintactico.java AnalizadorLexico.java
Utl.java Token.java EntradaSalida.java LexicoException.java
SintacticException.java Tipo.java . . . (todas las clases)
```

Para la ejecución del programa desde consola se deberá ejecutar

```
>java Minijavac <archivo fuente>
```

Donde <archivo fuente> es el path absoluto o relativo dependiendo de donde se encuentra el archivo fuente a analizar en el equipo.

Ejemplos:

```
>java Minijavac ./Test/Correctos/Test1.java
```

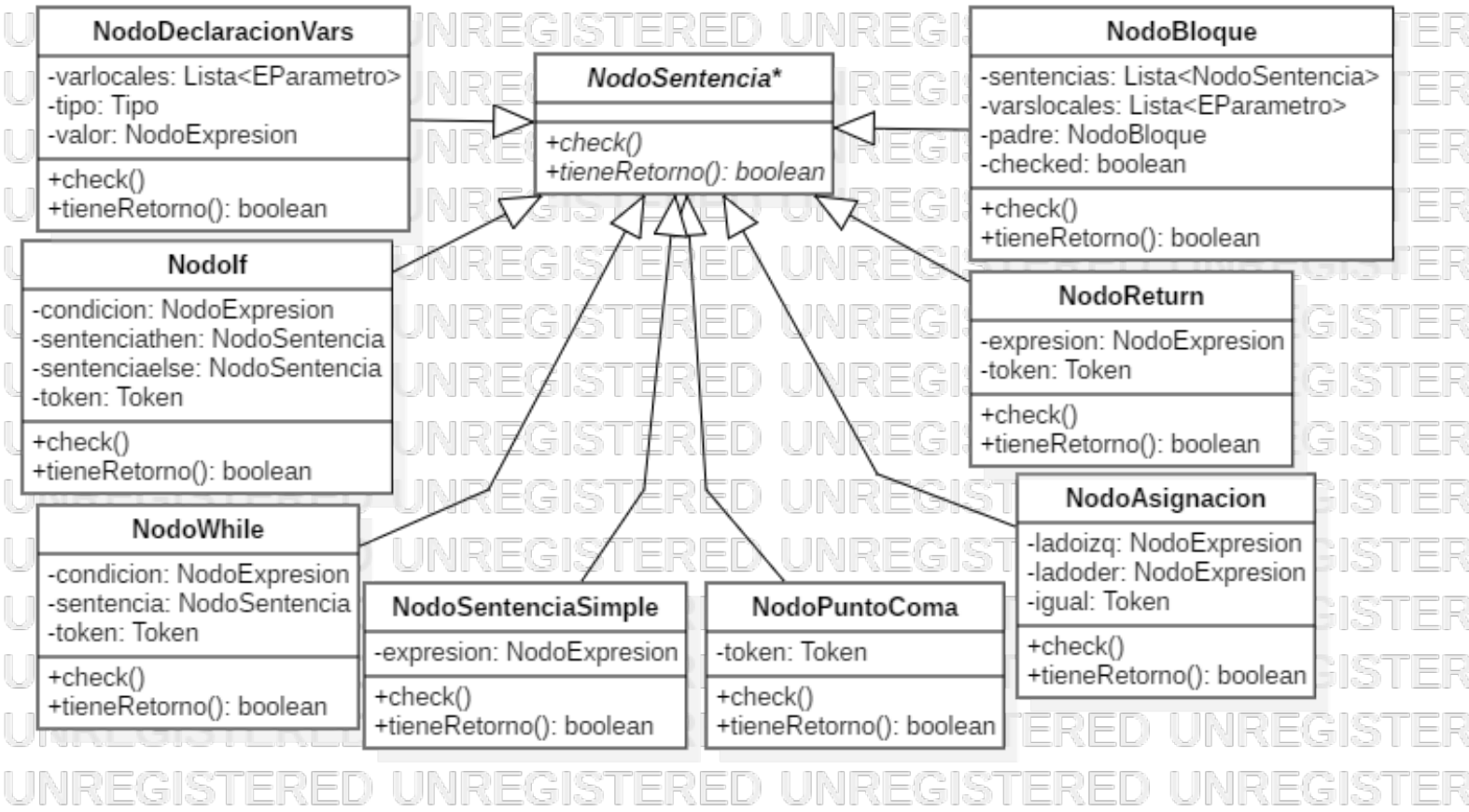
El archivo Test1.java debe estar dentro del directorio Test/Correctos/ que se encuentra en el mismo directorio que los archivos fuente

```
>java Minijavac C:/Testing/Incorrectos/Test2.java
```

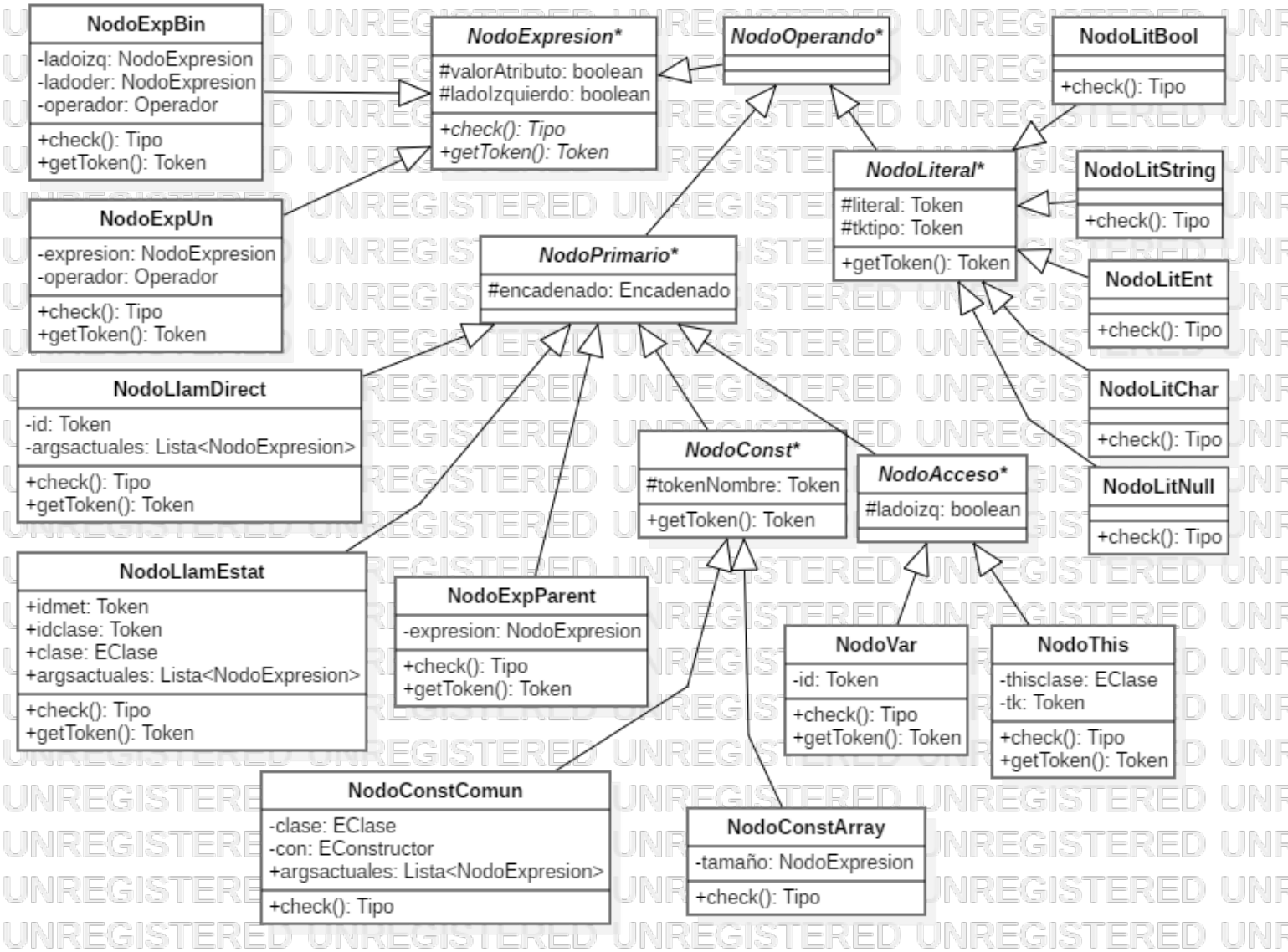
El archivo Test2.java debe estar dentro del directorio que especifica el path absoluto

Diagramas de clases del AST.

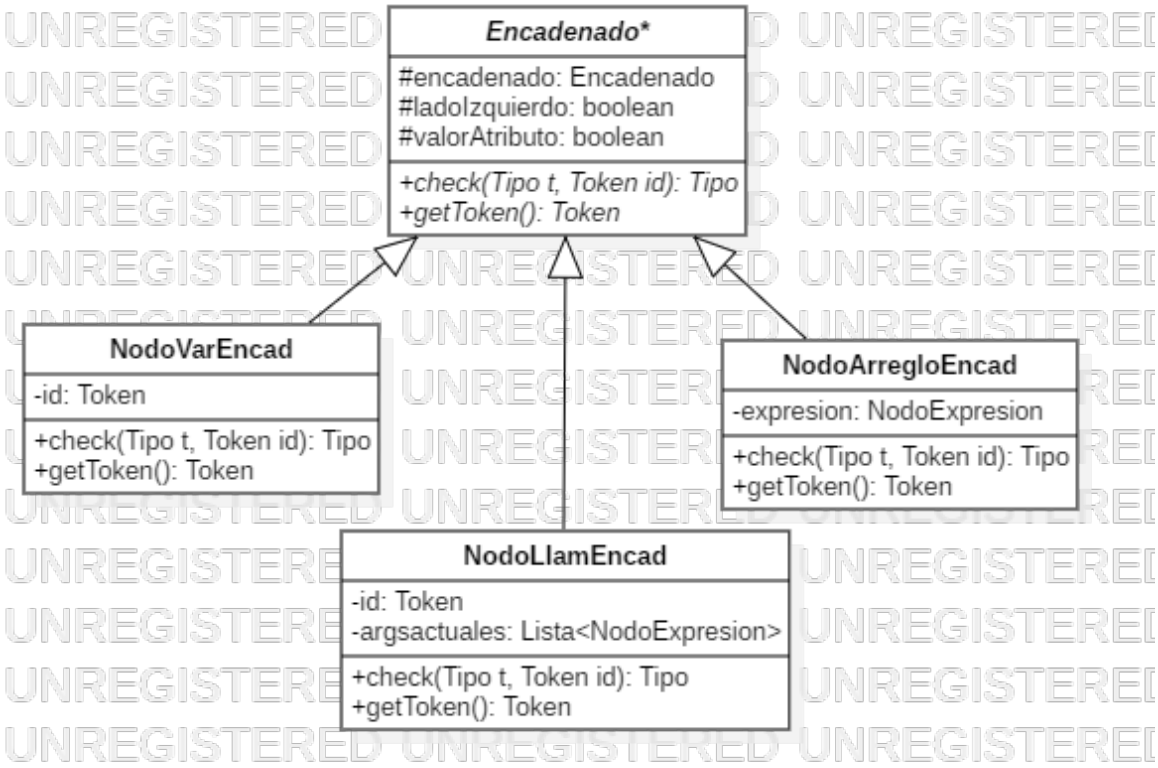
Sentencias:



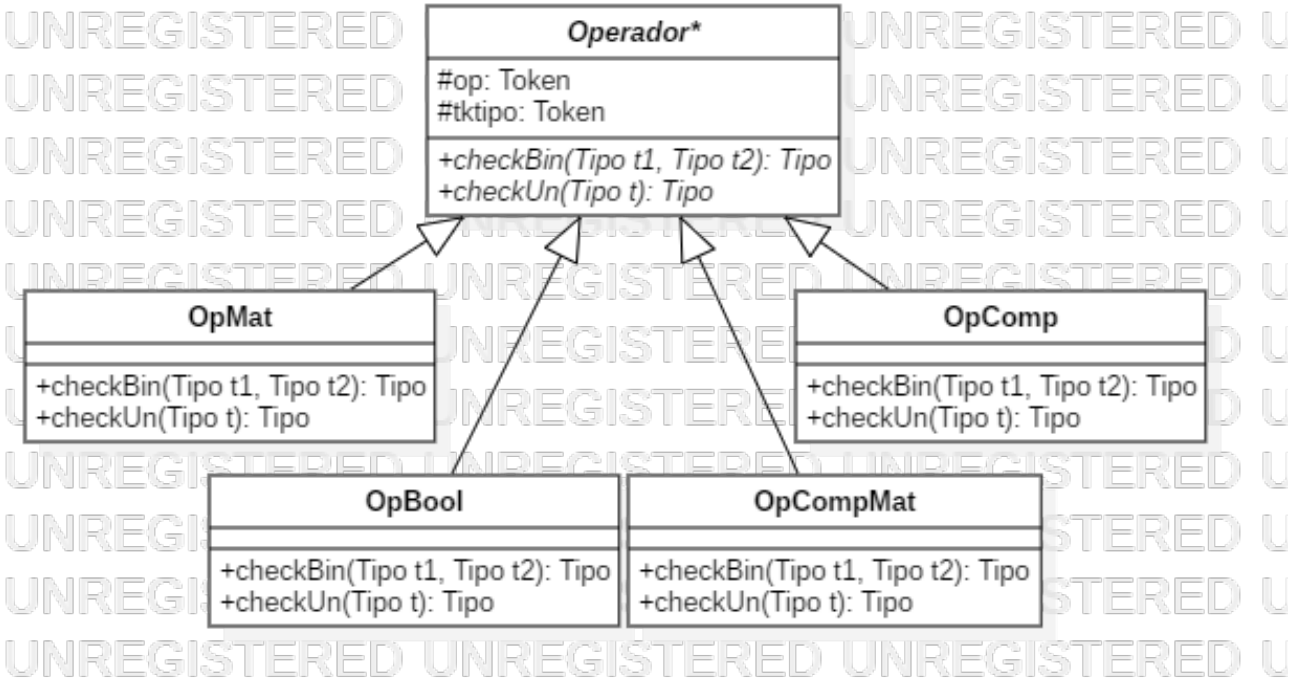
Expresiones:



Encadenados:



Operadores:



Esquema de traducción.

Inicio -> Clase Clases $
Clases -> Clase Clases
Clases -> ε
Clase -> class idClase Herencia { Miembros }
Herencia -> extends idClase
Herencia -> ε
Miembros -> Miembro Miembros
Miembros -> ε
Miembro -> Atributo
Miembro -> Ctor
Miembro -> Metodo

Atributo ->
	Visibilidad Tipo ListaDecVars
	Inicializacion
	{Se guarda el NodoExpresion que retorna inicializacion}
	{Si el NodoExpresion no es nulo se setea que es un valor de atributo}
	{Se crean los atributos en la clase actual con el NodoExpresion como valor}
	;

Inicializacion -> = Expresion {Se retorna el NodoExpresion que retorna Expresion}
Inicializacion -> ε {Se retorna null}

Metodo ->
	FormaMetodo TipoMetodo idMetVar ArgsFormales
	{Antes de llamar a bloque se setea al bloque actual como nulo}
	Bloque {Se guarda el NodoBloque que retorna Bloque}
	{Se setea el NodoBloque guardado al miembro actual}

Ctor ->
	idClase ArgsFormales
	{Antes de llamar a bloque se setea al bloque actual como nulo}
	Bloque {Se guarda el NodoBloque que retorna Bloque}
	{Se setea el NodoBloque guardado al miembro actual}

ArgsFormales -> (ListaArg)
ListaArg -> Arg ArgFormales
ListaArg -> ε
ArgFormales -> , Arg ArgFormales
ArgFormales -> ε
Arg -> Tipo idMetVar
FormaMetodo -> static
FormaMetodo -> dynamic
Visibilidad -> public
Visibilidad -> private
TipoMetodo -> Tipo
TipoMetodo -> void
Tipo -> boolean PosibleArreglo
Tipo -> char PosibleArreglo
Tipo -> int PosibleArreglo
Tipo -> idClase PosibleArreglo
Tipo -> String PosibleArreglo
PosibleArreglo -> [ ] PosibleArreglo
PosibleArreglo -> ε
ListaDecVars -> idMetVar ListaDV
ListaDV -> , idMetVar ListaDV
ListaDV -> ε

Bloque ->
	{
	{Se crea un nuevo NodoBloque con el bloque actual como padre}
	{Se setea al NodoBloque recien creado como bloque actual}
	Sentencias
	}
	{Se setea al padre del bloque actual, como bloque actual}
	{Se retorna el NodoBloque creado}

Sentencias ->
	Sentencia {Se guarda el NodoSentencia que retorna Sentencia}
	{Se agrega el NodoSentencia al bloque actual}
	Sentencias

Sentencias -> ε

Sentencia ->
	; {Se crea y se retorna un NodoPuntoComa}

Sentencia ->
	if ({Se crea un nuevo NodoIf con el token "if"}
	Expresion {Se setea el NodoExpresion que retorna Expresion como condicion del NodoIf}
	) Sentencia {Se setea el NodoSentencia que retorna Sentencia como sentencia then del NodoIf}
	SentenciaElse {Se setea el NodoSentencia que retorna Sentencia como sentencia else del NodoIf}
	{Se retorna el NodoIf creado}

Sentencia ->
	while ({Se crea un nuevo NodoWhile con el token "while"}
	Expresion {Se setea el NodoExpresion que retorna Expresion como condicion del NodoWhile}
	) Sentencia {Se setea el NodoSentencia que retorna Sentencia como sentencia del NodoWhile}
	{Se retorna el NodoWhile creado}

Sentencia ->
	return {Se crea un nuevo NodoReturn con el token "return"}
	Expresiones {Se setea el NodoExpresion que retorna Expresiones como expresion del NodoReturn}
	;
	{Se retorna el NodoReturn creado}

Sentencia ->
	Asignacion {Se guarda el NodoAsignacion que retorna Asignacion}
	;
	{Se retorna el nodo guardado}

Sentencia ->
	{Se crea un nuevo NodoSentenciaSimple}
	SentenciaLlamada {Se setea el NodoExpresion que retorna SentenciaLlamada como expresion del NodoSentenciaSimple}
	;
	{Se retorna el NodoSentenciaSimple creado}

Sentencia ->
	Tipo {Se guarda el Tipo que retorna}
	{Se crea una nueva lista de EParametro para pasarle a ListaDecVars}
	{Se llama a ListaDecVars con la lista, y el tipo guardado} ListaDecVars
	{Se crea un nuevo NodoDecVars con el Tipo y la lista}
	Inicializacion {Se setea el NodoExpresion que retorna Inicializacion como valor del NodoDecVars}
	;
	{Se retorna el NodoDecVars creado}

Sentencia -> Bloque {Se retorna el NodoBloque que retorna Bloque}

SentenciaElse -> else Sentencia {Se retorna el NodoSentencia que retorna Sentencia}
SentenciaElse -> ε {Se retorna null}

Expresiones -> Expresion {Se retorna el NodoExpresion que retorna Expresion}
Expresiones -> ε {Se retorna null}

Asignacion ->
	{Se crea un nuevo NodoAsignacion}
	AccesoVar {Se guarda el NodoVar que retorna}
	{Se setea que el NodoVar esta del lado izquierdo}
	{Se setea el NodoVar como lado izquierdo del NodoAsignacion}
	= {Se setea el token "=" en el NodoAsignacion}
	Expresion {Se setea el NodoExpresion que retorna Expresion como lado derecho del NodoAsignacion}
	{Se retorna el NodoAsignacion creado}

Asignacion ->
	{Se crea un nuevo NodoAsignacion}
	AccesoThis {Se guarda el NodoThis que retorna}
	{Se setea que el NodoThis esta del lado izquierdo}
	{Se setea el NodoThis como lado izquierdo del NodoAsignacion}
	= {Se setea el token "=" en el NodoAsignacion}
	Expresion {Se setea el NodoExpresion que retorna Expresion como lado derecho del NodoAsignacion}
	{Se retorna el NodoAsignacion creado}

SentenciaLlamada ->
	(Primario {Se guarda el NodoExpresion que retorna Primario}
	)
	{Se retorna el NodoExpresion guardado}

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

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

ExpOrR ->
	|| {Se crea un nuevo OpBool con el token}
	{Se crea un nuevo NodoExpBin con el OpBool y el NodoExpresion recibido como parametro como lado izquierdo}
	ExpAnd {Se setea el NodoExpresion que retorna como lado derecho del NodoExpBin}
	{Se llama a ExpOrR con el NodoExpBin creado} ExpOrR
	{Se retorna el NodoExpresion que retorna ExpOrR}

ExpOrR -> ε {Se retorna el NodoExpresion recibido como parametro}

ExpAnd ->
	ExpIg {Se guarda el NodoExpresion que retorna}
	{Se llama a ExpAndR con el nodo guardado} ExpAndR
	{Se retorna el NodoExpresion que retorna ExpAndR}

ExpAndR ->
	&& {Se crea un nuevo OpBool con el token}
	{Se crea un nuevo NodoExpBin con el OpBool y el NodoExpresion recibido como parametro como lado izquierdo}
	ExpIg {Se setea el NodoExpresion que retorna como lado derecho del NodoExpBin}
	{Se llama a ExpAndR con el NodoExpBin creado} ExpAndR
	{Se retorna el NodoExpresion que retorna ExpAndR}

ExpAndR -> ε {Se retorna el NodoExpresion recibido como parametro}

ExpIg ->
	ExpComp {Se guarda el NodoExpresion que retorna}
	{Se llama a ExpIgR con el nodo guardado} ExpIgR
	{Se retorna el NodoExpresion que retorna ExpIgR}

ExpIgR ->
	OpIgual {Se guarda el Operador que retorna}
	{Se crea un nuevo NodoExpBin con el Operador y el NodoExpresion recibido como parametro como lado izquierdo}
	ExpComp {Se setea el NodoExpresion que retorna como lado derecho del NodoExpBin}
	{Se llama a ExpIgR con el NodoExpBin creado} ExpIgR
	{Se retorna el NodoExpresion que retorna ExpIgR}

ExpIgR -> ε {Se retorna el NodoExpresion recibido como parametro}

ExpComp ->
	ExpAd {Se guarda el NodoExpresion que retorna}
	{Se llama a ExpCompR con el nodo guardado} ExpCompR
	{Se retorna el NodoExpresion que retorna ExpCompR}

ExpCompR ->
	OpComp {Se guarda el Operador que retorna}
	{Se crea un nuevo NodoExpBin con el Operador y el NodoExpresion recibido como parametro como lado izquierdo}
	ExpAd {Se setea el NodoExpresion que retorna como lado derecho del NodoExpBin}
	{Se retorna el NodoExpBin creado}

ExpCompR -> ε {Se retorna el NodoExpresion recibido como parametro}

ExpAd ->
	ExpMul {Se guarda el NodoExpresion que retorna}
	{Se llama a ExpAdR con el nodo guardado} ExpAdR
	{Se retorna el NodoExpresion que retorna ExpAdR}

ExpAdR ->
	OpAd {Se guarda el Operador que retorna}
	{Se crea un nuevo NodoExpBin con el Operador y el NodoExpresion recibido como parametro como lado izquierdo}
	ExpMul {Se setea el NodoExpresion que retorna como lado derecho del NodoExpBin}
	{Se llama a ExpAdR con el NodoExpBin creado} ExpAdR
	{Se retorna el NodoExpresion que retorna ExpAdR}

ExpAdR -> ε {Se retorna el NodoExpresion recibido como parametro}

ExpMul ->
	ExpUn {Se guarda el NodoExpresion que retorna}
	{Se llama a ExpMulR con el nodo guardado} ExpMulR
	{Se retorna el NodoExpresion que retorna ExpMulR}

ExpMulR ->
	OpMul {Se guarda el Operador que retorna}
	{Se crea un nuevo NodoExpBin con el Operador y el NodoExpresion recibido como parametro como lado izquierdo}
	ExpUn {Se setea el NodoExpresion que retorna como lado derecho del NodoExpBin}
	{Se llama a ExpMulR con el NodoExpBin creado} ExpMulR
	{Se retorna el NodoExpresion que retorna ExpMulR}

ExpMulR -> ε {Se retorna el NodoExpresion recibido como parametro}

ExpUn -> OpUn ExpUn
	OpUn {Se guarda el Operador que retorna}

{Se crea un nuevo NodoExpUn con el Operador}
ExpUn {Se setea el NodoExpresion que retorna como expresion del NodoExpBin}
{Se retorna el NodoExpUn creado}

ExpUn -> Operando {Se retorna el NodoExpresion que retorna Operando}

OpIgual -> == | != {Se crea y se retorna un OpComp con el token}

OpComp -> < | > | <= | >= {Se crea y se retorna un OpCompMat con el token}

OpAd -> + | - {Se crea y se retorna un OpMat con el token}

OpUn -> + | - {Se crea y se retorna un OpMat con el token}

OpUn -> ! {Se crea y se retorna un OpBool con el token}

OpMul -> * | / {Se crea y se retorna un OpMat con el token}

Operando -> Literal {Se retorna el NodoExpresion que retorna Literal}
Operando -> Primario {Se retorna el NodoExpresion que retorna Primario}

Literal -> null {Se crea y se retorna un NodoLitNull con el token}
Literal -> true {Se crea y se retorna un NodoLitBool con el token}
Literal -> false {Se crea y se retorna un NodoLitBool con el token}
Literal -> intLiteral {Se crea y se retorna un NodoLitInt con el token}
Literal -> charLiteral {Se crea y se retorna un NodoLitChar con el token}
Literal -> stringLiteral {Se crea y se retorna un NodoLitString con el token}

Primario ->
idMetVar {Se guarda el token}
{Se llama a MetodoVariable con el token} MetodoVariable
{Se retorna el NodoPrimario que retorna MetodoVariable}

Primario -> ExpresionParentizada {Se retorna el NodoPrimario que retorna ExpresionParentizada}

Primario -> AccesoThis {Se retorna el NodoPrimario que retorna AccesoThis}

Primario -> LlamadaMetodoEstatico {Se retorna el NodoPrimario que retorna LlamadaMetodoEstatico}

Primario -> LlamadaCtor {Se retorna el NodoPrimario que retorna LlamadaCtor}

MetodoVariable ->
{Se crea un nuevo NodoLlamadaDirecta con el token recibido como parametro}
{Se llama a ArgsActuales con la lista de argumentos del nodo creado}
ArgsActuales
Encadenado {Se setea el Encadenado que retorna como encadenado del NodoLlamadaDirecta}
{Se retorna el NodoLlamadaDirecta creado}
MetodoVariable ->
{Se crea un nuevo NodoVar con el token recibido como parametro}
Encadenado {Se setea el Encadenado que retorna como encadenado del NodoVar}
{Se retorna el NodoVar creado}

ExpresionParentizada ->
{Se crea un nuevo NodoExpParent}
(Expresion {Se setea al NodoExpresion retornado como expresion del NodoExpParent}
) Encadenado {Se setea al Encadenado retornado como encadenado del NodoExpParent}
{Se retorna el nodo creado}

Encadenado ->
. idMetVar {Se guarda el token idMetVar}
{Se llama a Acceso con el token guardado}
Acceso {Se retorna el Encadenado retornado por Acceso}

Encadenado -> AccesoArregloEncadenado {Se retorna el Encadenado retornado por AccesoArregloEncadenado}
Encadenado -> ε {Se retorna null}

Acceso ->
	{Se llama a LlamadaMetodoEncadenado con el token recibido como parametro}
	LlamadaMetodoEncadenado {Se retorna el Encadenado retornado por LlamadaMetodoEncadenado}
Acceso ->
	{Se llama a AccesoVarEncadenado con el token recibido como parametro}
	AccesoVarEncadenado {Se retorna el Encadenado retornado por AccesoVarEncadenado}

AccesoThis ->
	this {Se guarda el token}
	{Se crea un nuevo NodoThis con el token y la clase actual}
	Encadenado {Se setea el Encadenado retornado como encadenado del NodoThis}
	{Se retorna el NodoThis creado}

AccesoVar ->
	idMetVar {Se guarda el token}
	{Se crea un nuevo NodoVar con el token}
	Encadenado {Se setea el Encadenado retornado como encadenado del NodoVar}
	{Se retorna el NodoVar creado}

LlamadaMetodoEstatico ->
	idClase {Se guarda el token}
	.
	{Se llama a LlamadaMetodo con el token guardado]
	LlamadaMetodo {Se retorna el NodoLlamEstat que retorna LlamadaMetodo}

LlamadaMetodo ->
	idMetVar {Se guarda el token}
	{Se crea un nuevo NodoLlamEstat con el token guardado y el recibido como parametro}
	{Se llama a ArgsActuales con la lista de argumentos del nodo recien creado}
	ArgsActuales
	Encadenado {Se setea el Encadenado retornado como encadenado del NodoLlamEstat}
	{Se retorna el NodoLlamEstat creado}

LlamadaCtor -> new LlamadaCtorR {Se retorna el NodoConst que retorna LlamadaCtorR}

LlamadaCtorR ->
	idClase {Se guarda el token}
	{Se llama a LlamadaCtorIDClase con el token}
	LlamadaCtorIDClase {Se retorna el NodoConst que retorna LlamadaCtorIDClase}

LlamadaCtorR ->
	{Se crea un nuevo NodoConstArray con el token actual}
	TipoArreglo
	[Expresion {Se setea al NodoExpresion retornado como tamaño del NodoConstArray}
	]
	Encadenado {Se setea al Encadenado retornado como encadenado del NodoConstArray}
	{Se retorna el NodoConstArray creado}

LlamadaCtorIDClase ->
	{Se crea un nuevo NodoConstArray con el token recibido como parametro}
	[Expresion {Se setea al NodoExpresion retornado como tamaño del NodoConstArray}
	] Encadenado {Se setea al Encadenado retornado como encadenado del NodoConstArray}
	{Se retorna el NodoConstArray creado}

LlamadaCtorIDClase ->
	{Se crea un nuevo NodoConstComun con el token recibido como parametro}
	{Se llama a ArgsActuales con la lista de argumentos vacia del nodo recien creado}

ArgsActuales
Encadenado {Se setea al Encadenado retornado como encadenado del NodoConstComun}
{Se retorna el NodoConstComun creado}

TipoArreglo -> char
TipoArreglo -> boolean
TipoArreglo -> int
TipoArreglo -> String
ArgsActuales -> (ListaExpresiones)
ListaExpresiones -> Expresion ListaExp
ListaExpresiones -> ε
ListaExp -> , Expresion ListaExp
ListaExp -> ε

LlamadaMetodoEncadenado ->
{Se crea un nuevo NodoLlamEncad con el token recibido como parametro}
{Se llama a ArgsActuales con la lista de argumentos vacia del nodo recien creado}
ArgsActuales
Encadenado {Se setea al Encadenado retornado como encadenado del NodoLlamEncad}
{Se retorna el NodoLlamEncad creado}

AccesoVarEncadenado ->
{Se crea un nuevo NodoVarEncad con el token recibido como parametro}
Encadenado {Se setea al Encadenado retornado como encadenado del NodoVarEncad}
{Se retorna el NodoVarEncad creado}

AccesoArregloEncadenado ->
{Se crea un nuevo NodoArregloEncad}
[Expresion {Se setea al NodoExpresion retornado como expresion del NodoArregloEncad}
] Encadenado {Se setea al Encadenado retornado como encadenado del NodoArregloEncad}
{Se retorna el NodoArregloEncad creado}

Chequeos semánticos realizados y errores detectados.

A continuación explicaran brevemente los chequeos semánticos para cada tipo de nodo del AST.

Chequeos de sentencias:

NodoBloque
Si el bloque no fue chequeado:
- Asignar como bloque actual
- Recorrer la lista de sentencias invocando el chequeo de cada una
- Desapilar del miembro actual las variables locales del bloque
- Asignar como bloque actual al bloque padre
- Setear que el bloque fue chequeado

NodoDeclaracionVars
Primero chequear que el tipo este definido, si no lo esta reportar error
Si tiene expresion valor, chequear que los tipos sean compatibles sino reportar error
Si no es una sentencia unica (dentro de un if o un while)
- Recorrer la lista de variables y agregarlas al miembro actual, si hay variables con mismo nombre reportar error
- Si no hay error, agregar las variables al bloque actual

NodoIf
Chequear que la condicion sea de tipo booleano, sino reportar error
Chequear la sentencia del then
Si tiene sentencia else, chequear la sentencia else

NodoWhile
Chequear que la condicion sea de tipo booleano, sino reportar error
Chequear la sentencia.

NodoReturn
Si el miembro actual no es un constructores
- Obtener el tipo de retorno del miembros
- Si la expresion no es nula
  - Chequearla y comparar con el tipo de retorno, si no son compatibles reportar error
- Si la expresion es nula y el tipo de retorno no es void reportar error

NodoSentenciaSimple
Chequear la expresion

NodoPuntoComa
No hay nada que chequear

NodoAsignacion
Chequear el lado izquierdo y obtener su tipo
Chequear el lado derecho y obtener su tipo
Si los tipos no son compatibles reportar error

Chequeos de expresiones:

NodoExpBin
Chequear el lado izquierdo y obtener su tipo
Chequear el lado derecho y obtener su tipo
Utilizar el chequeo binario del operador con los tipos obtenidos
El tipo resultante es que retorna el operador

NodoExpUn
Chequear la expresion y obtener su tipo
Utilizar el chequeo unario del operador con el tipo obtenido
El tipo resultante es que retorna el operador

NodoLlamDirect
Chequear que el metodo sea visible desde la clase actual, si no lo es reportar error
Obtener la forma del metodo
Si la llamada no esta en la inicializacion de un atributo
    Obtener la forma del miembro actual
    Si el metodo es estatico ó el miembro actual es dinamico
        Chequear que los argumentos sean correctos (cantidad, orden y tipo), si no lo son reportar error
        Obtener el tipo de retorno del metodo.
        Si la llamada tiene encadenado
            Chequear el encadenado pasandole el tipo de retorno del metodo y retornar el tipo devuelto
        Si no
            Retornar el tipo de retorno del metodo

    Sino, reportar error de intento de acceso a metodo dinamico desde un metodo estatico
Si la llamada esta en una inicializacion de atributo
    Si el metodo es estatico
        Chequear argumentos, reportando error, obtener tipo retorno del metodo y retornar el tipo dependiendo si tiene encadenado o no
    Si el metodo es dinamico, reportar error

NodoLlamEstat
Primero chequeo que la clase este definida, sino reportar error
Chequeo que el metodo exista en la clase, sino reportar error

Luego chequeo que los parametros sean correctos (cantidad, orden y tipos), si no lo son reportar error
Obtener el tipo de retorno del metodo
Por ultimo si hay encadenado, chequear el encadenado pasandole como parametro el tipo de retorno y retornar el tipo que devuelve
Sino hay encadenado retornar el tipo de retorno del metodo

NodoExpParent
Chequear la expresion y obtener su tipo
Si hay encadenado, chequear el encadenado pasandole como parametro el tipo de la expresion y retornar el tipo que devuelve.
Sino, retornar el tipo de la expresion

NodoConstArray
Chequear el tamaño y obtener su tipo
Chequear que el tipo de tamaño sea entero, si no lo es reportar error
Si el tipo del arreglo es de tipo clase, chequear que la clase este definida, sino reportar error
Luego, si hay encadenado, chequear el encadenado pasandole como parametro el tipo del arreglo y retornar el tipo que devuelve.
Si no hay encadenado, retornar el tipo del arreglo

NodoConstComun
Primero chequear que la clase este definida, sino reportar error
Luego chequear que los parametros sean correctos (cantidad, orden y tipo) con alguno de los constructores de la clase chequeando cada argumentos, si no son correctos reportar error
Obtener el tipo de retorno del constructor.
Por ultimo, si hay encadenado, chequear el encadenado pasandole como parametro el tipo de retorno y retornar el tipo que devuelve.
Si no hay encadenado, retornar el tipo de retorno del constructor.

NodoVar
Si esta en una inicializacion de atributos, reportar error
Sino, primero chequear si la variable es una variable local o parametro
Si no esta
    Si el miembro actual es estatico, reportar error de acceso a variable a instancia.
    Si no, chequear si es un atributo visible de la clase actual y si no lo es por ultimo reportar error
Luego obtener el tipo de la variable

Por ultimo, si hay encadenado, chequear el encadenado pasandole como parametro el tipo de la variable y retornar el tipo que devuelve.
Si no hay encadenado, retornar el tipo de la variable.

NodoThis
Si esta en una inicializacion de atributo, reportar error
Si no esta

> Si hay encadenado, chequear el encadenado pasandole como parametro el tipo de la clase y retornar el tipo que devuelve.
> Si no hay encadenado, retornar el tipo de la clase.

NodoLitInt
NodoLitBool
NodoLitChar
NodoLitString
NodoLitNull
Para todos los literales simplemente se retorna el tipo del literal

Chequeos de expresiones:

NodoVarEncad
Primero chequear que el tipo recibido sea de tipo clase, si no lo es reportar error
Luego chequear que la clase este definida, sino reportar error
Chequear que la clase tenga un atributo visible desde la clase actual, sino reportar error
Obtener el tipo del atributo
Finalmente, si hay encadenado, chequear el encadenado pasandole como parametro el tipo del atributo y retornar el tipo que devuelve.
Si no hay encadenado, retornar el tipo del atributo.

NodoLlamEncad
Primero chequear que el tipo recibido sea de tipo clase, si no lo es reportar error
Luego chequear que la clase este definida, sino reportar error
Chequeo que la clase un metodo con el mismo nombre, sino reportar error
Chequear que los parametros sean correctos (cantidad, orden y tipo) chequeando cada argumento, si no son correctos reportar error
Obtener el tipo de retorno del metodo
Finalmente, si hay encadenado, chequear el encadenado pasandole como parametro el tipo de retorno del metodo y retornar el tipo que devuelve.
Si no hay encadenado y se esta en un lado izquierdo de una asignacion, repotar error
Sino, retornar el tipo de retorno del metodo

NodoArregloEncad
Primero chequear que el tipo recibido sea un arreglo, si no lo es reportar error
Luego chequear la expresion y obtener el tipo
Chequear que el tipo de la expresion sea entero, si no lo es reportar error
Finalmente, si hay encadenado, chequear el encadenado pasandole como parametro el tipo recibido como parametro y retornar el tipo que devuelve.
Si no hay encadenado, retornar el mismo tipo que el recibido pero que no sea arreglo.

Chequeos de operadores:

OpMat
Siempre se chequea que los tipos recibidos como parametros para los chequeos binarios y unarios sean de tipo entero
El tipo que retorna es entero.

OpBool
Siempre se chequea que los tipos recibidos como parametros para los chequeos binarios y unarios sean de tipo booleano
El tipo que retorna es booleano.

OpCompMat
Siempre se chequea que los tipos recibidos como parametros para el chequeos binarios sean de tipo entero. No se utiliza el chequeo unario.
El tipo que retorna es booleano.

OpComp
Se chequean que los tipos recibidos como parametros para el chequeo binario sean compatibles hacia ambos lados. No se utiliza el chequeo unario.
El tipo que retorna es booleano.

Testing

Para el testing del programa se utilizaron varios casos de prueba tanto correctos como incorrectos.
Cada caso de test tanto correcto como incorrecto contiene al principio un comentario con la explicación, para la cual esta hecho dicho caso.