Proyecto: Etapa 4

MiniJava 2018: Compiladores e Intérpretes

Germán Alejandro Gómez

23/10/2018

INDICE

Instrucciones para ejecutar.

Para poder utilizar el programa hace falta seguir las siguientes instrucciones:

1. Abrir una consola en Windows, por ejemplo cmd.
2. Usando cmd, ir al directorio donde se encuentra Principal.java.
3. Utilizar el comando 'javac Principal.java' para compilar el código.
4. Para ejecutarlo hay que ingresar la siguiente sentencia: 'java Principal <IN_FILE>'. IN_FILE es el archivo a compilar. Cualquier error léxico o sintáctica se mostrará por consola.

Casos de prueba

Se adjunta una carpeta con el nombre test. Cada archivo corresponde a un caso de prueba, y el resultado esperado, se especifica en forma de comentario dentro de cada uno.

EDT

<Empezar> ::= { creo la tabla de símbolos } <Inicial> **eof**

<Inicial> ::= <Clase><MasClase>

<MasClase>::= <Clase><MasClase> | ε

<Clase> ::= **class idClase**{ creo una EntradaClase con el token IdClase y lo agrego a TS} <Herencia> **{** <Miembro>**}** { Agrego un constructor con 0 parámetros si no tiene constructor EntradaClase}

<Miembro> ::= <Atributo><Miembro> |

<Ctor> { agrego <Ctor>.clase a la EntradaClase actual } <Miembro> |

<Metodo><Miembro> | ε

<Herencia> ::= ε { herencia de clase actual es 'Object'} | **extends idClase** { herencia de clase actual es IdClase.lex }

<Atributo> ::= <Visibilidad>{ <visibilidad>.tipo = <visibilidad>.tipo } <Tipo> >{ <tipo>.tipo = <tipo>.tipo } <ListaDecVars> { guardarAtributos(<listaDecVars>.atributos) en ClaseActual } <InLine> **;**

<InLine> ::= = <Expresion> {retorno el nodo expresion} | ε {retorno un nuevo nodo expresión vacio}

<Metodo> ::= <FormaMetodo><TipoMetodo>idMetVar {creo EntradaConParams con IdMetVar, <FormaMetodo>.tipo , <TipoMetodo>.dinamico } <ArgsFormales> { agrego EntradaMetodo a la clase Actual } <Bloque> {guardo el bloque que recibo en el bloque del metodo}

<Ctor> ::=**idClase** { <ctor>.clase = new EntradaClase(IdClase) } <ArgsFormales> { <ctor>.clase.guardar(<ArgsFormales>.paramatros) } <Bloque> {guardo el bloque que recibo en el bloque del metodo}

<ArgsFormales> ::= **(** <ListaArgsFormalesAux> { <ArgsFormales>.parametros = <listaArgsFormalesAux>.parametros } **)**

<ListaArgsFormalesAux> ::= <ListaArgsFormales> { <ListaArgsFormalesAux>.parametros = <listaArgsFormales>.parametros } | ε

<ListaArgsFormales> ::= <ArgFormal> { agrego <ArgFormal>.param a EntradaConParams actual } <F1>

<F1> ::= **,**<ListaArgsFormales> | ε

<ArgFormal> ::= <Tipo>**idMetVar** { <ArgFormal>.param = nuevo EntradaParametro con <Tipo>.tipo y token IdMetVar }

<FormaMetodo> ::= **static** { <formaMetodo>.dinamico = false} | **dynamic** {<formaMetodo>.dinamico = true }

<Visibilidad> ::= **public** {<visibilidad>.tipo = public} |**prívate** {<visibilidad>.tipo = private}

<ListaDecVars> ::= **idMetVar** { creo una EntradaAtributo con el token IdMetVar, <visibilidad>.tipo y <tipo>.tipo y lo agrego a ClaseActual } <F2>

<F2> ::= **,**<ListaDecVars> | ε

<TipoMetodo> ::= <Tipo> {tipo = <Tipo>.tipo} | **void** {tipo = void}

<Tipo> ::= **Boolean** {<Tipo>.tipo = boolean } <F10> {<Tipo>.tipo = <f10>.tipo } | **char** {<Tipo>.tipo = char } <F10> > {<Tipo>.tipo = <f10>.tipo } | **int** {<Tipo>.tipo = int } <F10> > {<Tipo>.tipo = <f10>.tipo } | **idClase** {<Tipo>.tipo = IdClase } | **String** {<Tipo>.tipo = String }

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

<F10> ::= **[]** {<f10>.tipo = TipoArreglo } | ε {<f10>.tipo = <Tipo>.tipo }

<Bloque> ::= **{** {recibo el bloque padre (null si es método el padre) **}** <MasSentencia>**}** {retorno el bloque que cree}

<MasSentencia> ::= <Sentencia> {recibo una lista de sentencias y la agrego a las sentencias ya encontradas} <MasSentencia> { le pido a masSentencias si hay mas sentecias} | ε {retorno la lista de sentencias}

<Sentencia> ::= **;** {retorno una sentencia vacia} |

<Asignacion>**;** {retorno una asignación}|

<SentenciaLlamada>**;** {retorno una sentencia llamada} |

<Tipo><ListaDecVars ><InLine>**;** {retorno una lista de declaración de variables y de asignaciones} |

if(<Expresion>**)** <Sentencia> {creo un nodo IfSinElse con expresión y sentencia } <F3> {retorno el NIf que retorna f3} |

while (<Expresion> **)** <Sentencia> { retorna un nodo while con espresion y sentencia } |

<Bloque> { retorno un bloque nuevo } |

return <ExpresionOpcional>**;** {retorno un nodo return}

<F3> ::= **else**<Sentencia> {retornar un NIfConElse } | ε {retorna NIfSinElse que recibió por parametro}

<Asignacion> ::= <AccesoVar>=<Expresion> {retorno nodo asignación con accesovar, igual y expresion} | <AccesoThis>= <Expresion> {retorno nodo asignación con accesothis, igual y expresion}

<SentenciaLlamada> ::= **(** <Primario> **)** {retorno nodo sentencia llamada con primario }

<ExpresionOpcional> ::= <ExpOr> {retorno el nodo expresión que retorna expOr } | ε {retorno ExpresionVacia}

<Expresion> ::= <ExpOr> {retorno el nodo expresión que retorna expOr }

<ExpOr> ::=<ExpAnd><ExpOr2> {retorno el nodo expresión que resulta de calcular ExpOr2 con el Nexpresion de ExpAnd}

<ExpOr2> ::= **||** <ExpAnd><ExpOr2> {retorno el nodo expresión que resulta de calcular ExpOr2 con el Nexpresion creada con la expresión pasada por parámetros y ExpAnd} | ε { retorna el nodo que recibe por parametro}

<ExpAnd> ::=<ExpIg><ExpAnd2> {retorno el nodo expresión que resulta de calcular ExpOr2 con el Nexpresion de ExpIg}

<ExpAnd2> ::=**&&**<ExpIg><ExpAnd2> {retorno el nodo expresión que resulta de calcular ExpAnd2 con el NExpresion creado con la expresión pasada por parámetros y ExpIg } | ε { retorna el nodo que recibe por parametro}

<ExpIg> ::=<ExpComp><ExpIg2> {retorno el nodo expresión que resulta de calcular ExpIg2 con el Nexpresion de ExpComp}

<ExpIg2> ::=<OpIg><ExpComp><ExpIg2> { retorno el nodo expresión que resulta de calcular ExpIg2 con el NExpresion creado con la expresión pasada por parámetros y ExpComp } | ε { retorna el nodo que recibe por parametro}

<ExpComp> ::=<ExpAd ><F8> {retorno el nodo expresión que resulta de calcular f8 con el Nexpresion de ExpAd}

<F8> ::= <OpComp><ExpAd> { retorno el nodo expresión creado con la expresión pasada por parámetros y ExpAd } | ε { retorna el nodo que recibe por parametro}

<ExpAd> ::=<ExpMul><ExpAd2> {retorno el nodo expresión que resulta de calcular ExpAd2 con el Nexpresion de ExpMul}

<ExpAd2> ::=<OpAd><ExpMul><ExpAd2> { retorno el nodo expresión que resulta de calcular ExpAd2 con el NExpresion creado con la expresión pasada por parámetros y ExpMul } | ε { retorna el nodo que recibe por parámetro}

<ExpMul> ::=<ExpUn><ExpMul2> {retorno el nodo expresión que resulta de calcular ExpMul2 con el Nexpresion de ExpUn}

<ExpMul2> ::=<OpMul><ExpUn><ExpMul2> { retorno el nodo expresión que resulta de calcular ExpMul2 con el NExpresion creado con la expresión pasada por parámetros y ExpUn } | ε { retorna el nodo que recibe por parámetro}

<ExpUn> ::= <OpUn><ExpUn> { retorno el nodo expresión unario creado con el resultado de ExpUn } | <Operando> { retorna el nodo expresión de operando }

<OpIg> ::= **==** | **!=**

<OpComp> ::= **<** | **>** | **<=** | **>=**

<OpAd> ::= **+** | **-**

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

<OpMul> ::= ***** | **/**

<Operando> ::= <Literal> { retorno un nodo literal que contiene el tipo como atr. } | <Primario> {retorno un nodo primario}

<Literal> ::=**null** | **true** | f**alse** | **entero**| c**aracter**| **string** {para todas las opciones de <literal>: retorna un nodo literal con el tipo al que representa }

<Primario> ::= <ExpresionParentizada> { retorna una expresión parentizada } | <AccesoThis> { retorna un acceso this } | **idMetVar** <F4> { retorna el resultado de <F4>} | <LlamadaMetodoEstatico> { retorna una llamada método estatico>}

| **new** <F9> {retorna el resultado de f9 }

<F4> ::= <ArgsActuales><Encadenado> {retorna una llamada método} | <Encadenado> { retorna un acceso var }

<LlamadaMetodo> ::=**idMetVar**<ArgsActuales><Encadenado> {retorna una llamada método}

<AccesoThis> ::=**this**<Encadenado> { retorna el nuevo nodo acceso this con encadenado y boolean li lugar (si esta a la izquierda del '=') }

<AccesoVar> ::= **idMetVar**<Encadenado> { retorna el nuevo nodo acceso var con encadenado y boolean li lugar (si esta a la izquierda del '=') }

<ExpresionParentizada> ::= **(** <Expresion> **)** <Encadenado> {retorna una expresión parentizada}

<LlamadaMetodoEstatico> ::= **idClase .** <LlamadaMetodo> {retorna una llamada método estático}

<F9> ::= **idClase**<ArgsActuales><Encadenado> { retorna una llamada Constructor de clase } | <TipoPrimitivo>**[** <Expresion>**]** <Encadenado> { retorna una llamada Constructor de arreglo }

<ArgsActuales> ::= (<ArgsActualesAux>) { retorna una clase con una lista de expresiones}

<ArgsActualesAux>::= <ListaExp> {devuelve lista de expresiones después de encontrar la ultima } | ε {devuelve una lista de expresiones }

<ListaExps> ::= <Expresion> <F5> {Encuentra una expresión, la agrega a la lista y le pide a f5 si hay alguna más}

<F5> ::= **,**<ListaExps> {Encuentra una coma por lo cual hay otro argumento, retorna la lista final de argumento que retorna listaExps} | ε { Retorna una lista de argumentos }

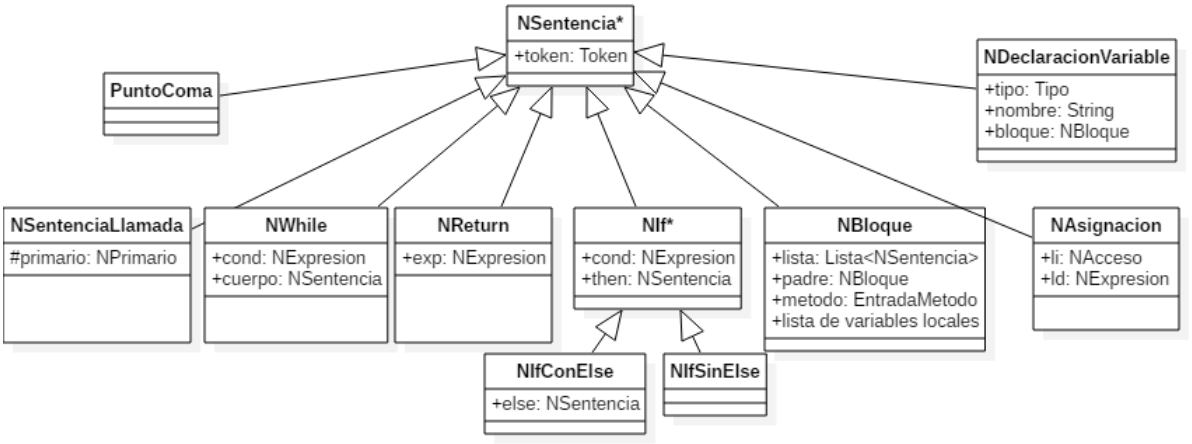
<Encadenado> ::= **.**<F6> {Retorna el resultado de f6}| <AccesoArregloEncadenado> {Retorna un acceso arreglo encadenado} | ε {retorna un encadenado vacio}

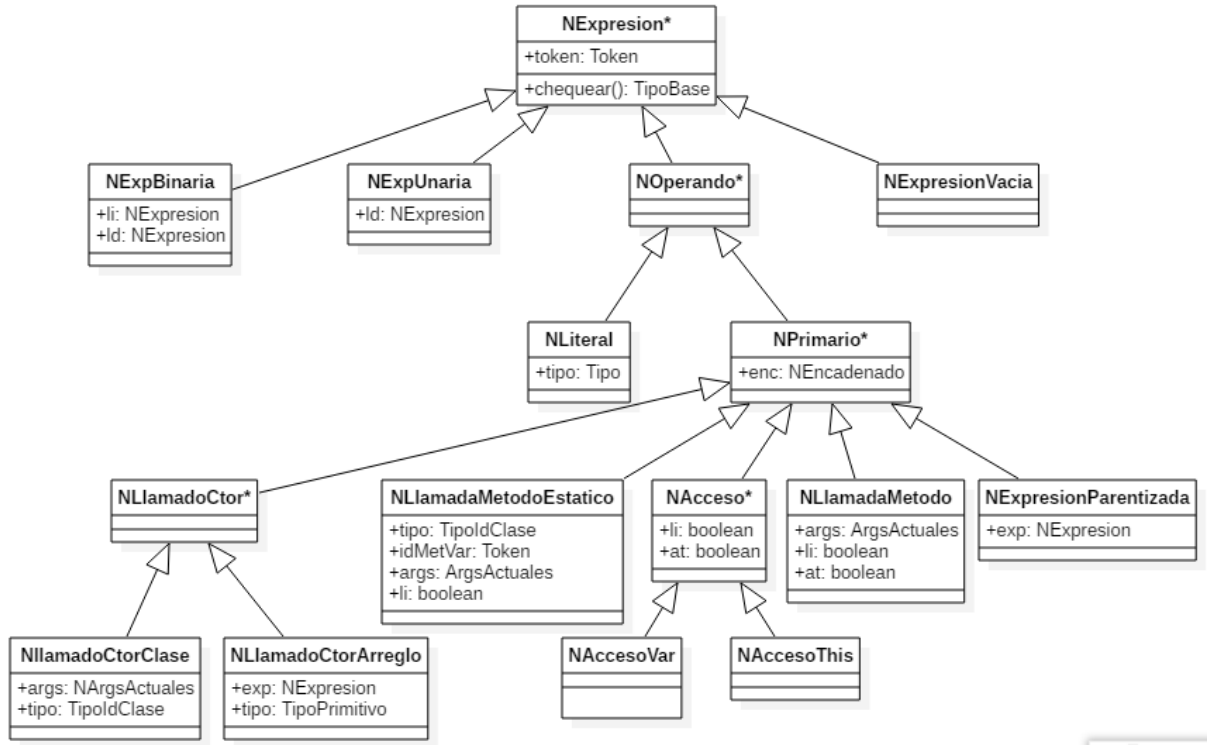
<F6> ::= **idMetVar** <F4> {Retorna el resultado de f7}

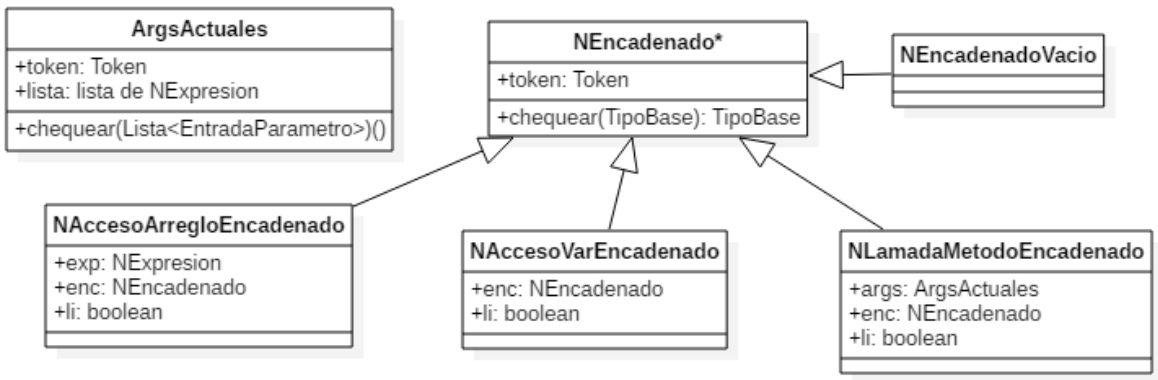
<F7> ::= <ArgsActuales><Encadenado> { retorna una llamada a un método encadenado } | <Encadenado> { retorna una llamada a una variable encadenada }

<AccesoArregloEncadenado> ::= **[**<Expresion>**]**<Encadenado> {Retorna un acceso arreglo encadenado}

Diagramas de clases







Consideraciones de diseño

- Se corrigieron los errores especificados de la etapa 3 que correspondía al logro recuperación de errores de declaración.
- La EDT no utiliza el método formal.
- La recuperación de errores de declaración no permite que se continúe con el chequeo de sentencias. Por ejemplo, si se encuentra un error semántico de declaración y además había un error semántico de sentencia, solo encontrará los errores de declaración.
- Cada chequear() de cada nodo del AST, especifica en forma de comentarios que tipos de chequeos se realizaron.

Logros

Se intentan cumplir los logros de:

- Entrega anticipada (etapa 4)
- Inicializaciones Inline controladas (etapa 4)
- Llamadas Sobrecargadas (etapa 4)
- Recuperación errores de declaración (etapa 3)
- Recuperación errores de sentencias (etapa 4)