Proyecto: Etapa 3

Compiladores e Intérpretes

Germán Alejandro Gómez

05/10/2018

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 subdividida en test sintácticos y test semánticos. 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> | ε

<Metodo> ::= <FormaMetodo><TipoMetodo>idMetVar {creo EntradaConParams con IdMetVar, <FormaMetodo>.tipo , <TipoMetodo>.dinamico } <ArgsFormales> { agrego EntradaMetodo a la clase Actual } <Bloque>

<Ctor> ::=**idClase** { <ctor>.clase = new EntradaClase(IdClase) } <ArgsFormales> { <ctor>.clase.guardar(<ArgsFormales>.paramatros) } <Bloque>

<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> ::= **{**<MasSentencia>**}**

<MasSentencia> ::= <Sentencia><MasSentencia> | ε

<Sentencia> ::= **;** | <Asignacion>**;** | <SentenciaLlamada>**;** | <Tipo><ListaDecVars ><InLine>**;** |

if(<Expresion> **)** <Sentencia> <F3> | **while (** <Expresion> **)** <Sentencia> | <Bloque> |

return <ExpresionOpcional>**;**

<F3> ::= **else**<Sentencia> | ε

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

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

<ExpresionOpcional> ::= <ExpOr> | ε

<Expresion> ::= <ExpOr>

<ExpOr> ::=<ExpAnd><ExpOr2>

<ExpOr2> ::= **||** <ExpAnd><ExpOr2> | ε

<ExpAnd> ::=<ExpIg><ExpAnd2>

<ExpAnd2> ::=**&&**<ExpIg><ExpAnd2> | ε

<ExpIg> ::=<ExpComp><ExpIg2>

<ExpIg2> ::=<OpIg><ExpComp><ExpIg2> | ε

<ExpComp> ::=<ExpAd ><F8>

<F8> ::= <OpComp><ExpAd> | ε

<ExpAd> ::=<ExpMul><ExpAd2>

<ExpAd2> ::=<OpAd><ExpMul><ExpAd2> | ε

<ExpMul> ::=<ExpUn><ExpMul2>

<ExpMul2> ::=<OpMul><ExpUn><ExpMul2> | ε

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

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

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

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

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

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

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

<Literal> ::=**null** | **true** | f**alse** | **entero**| c**aracter**| **string**

<Primario> ::= <ExpresionParentizada> | <AccesoThis> | **idMetVar** <F4> | <LlamadaMetodoEstatico>

| **new** <F9>

<F4> ::= <ArgsActuales><Encadenado> | <Encadenado>

<LlamadaMetodo> ::=**idMetVar**<ArgsActuales><Encadenado>

<AccesoThis> ::=**this**<Encadenado>

<AccesoVar> ::= **idMetVar**<Encadenado>

<ExpresionParentizada> ::= **(** <Expresion> **)** <Encadenado>

<LlamadaMetodoEstatico> ::= **idClase .** <LlamadaMetodo>

<F9> ::= **idClase**<ArgsActuales><Encadenado> | <TipoPrimitivo>**[** <Expresion>**]** <Encadenado>

<ArgsActuales> ::= (<ArgsActualesAux>)

<ArgsActualesAux>::= <ListaExp> | ε

<ListaExps> ::= <Expresion> <F5>

<F5> ::= **,**<ListaExps> | ε

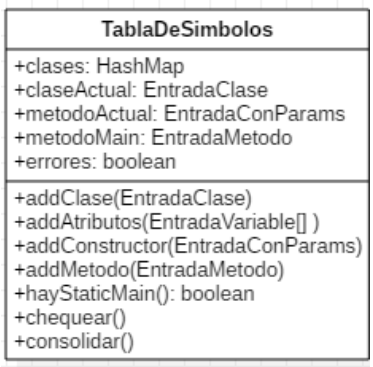
<Encadenado> ::= **.**<F6> | <AccesoArregloEncadenado> | ε

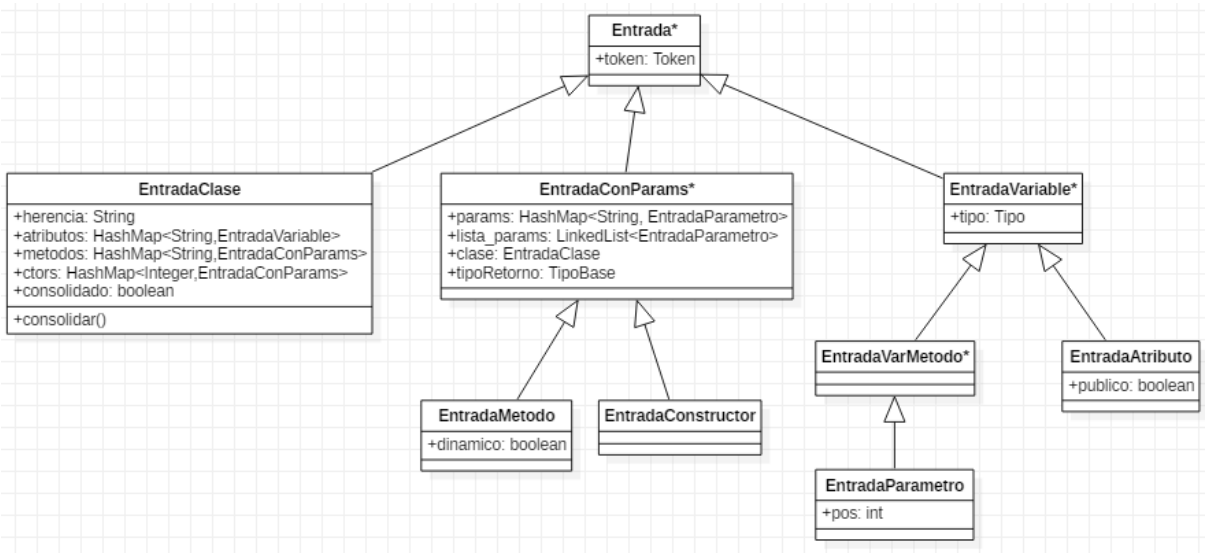
<F6> ::= **idMetVar** <F7>

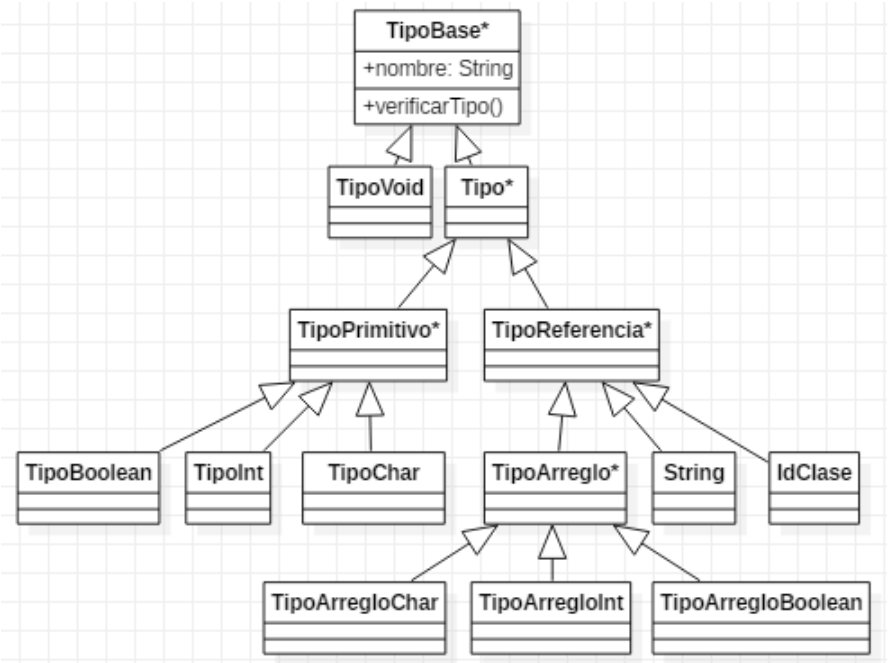
<F7> ::= <ArgsActuales><Encadenado> | <Encadenado>

<AccesoArregloEncadenado> ::= **[**<Expresion>**]**<Encadenado>

Diagramas de clases







Consideraciones de diseño

- Se corrigieron los errores especificados de la etapa 2.
- La EDT no utiliza el método formal.
- Solo se permite un static void main() por archivo de minijava. Si el programador decide poner un dynamic void main() en un clase, le quita la posibilidad de agregar un static void main() y no arrojara error.
- Si al momento de chequear que no haya herencia circular se encuentra con un caso de este tipo, la tabla de símbolos (TS) consolida la clase que hereda de la clase que genera conflicto. Por ejemplo si tenemos Clase1 extends Clase2, Clase2 extends Clase3, Clase3 extends Clase1, el compilador detectarte la herencia circular en Clase3, por lo cual lo marca como consolidado para que pueda seguir buscando otros errores. De igual manera ocurre que si por ejemplo, Clase1 extends Clase, y Clase no existe, el compilador muestra el error de que Clase no existe y a su vez consolida Clase1 para poder seguir consolidando.
- Los constructores se guardan en tablas dentro de cada EntradaClase. Su clave es representada por la cantidad de parámetros que poseen.
- Los métodos se guardan en tablas y listas dentro de cada EntradaClase. La clave en las tablas es generada a través de la concatenación del nombre del método, seguido de $ y la cantidad de parámetros. Por ejemplo ‘static void m1(int a)’ se almacena como ‘m1$1’.
- En la clase TablaDeSimbolos se encuentra el método chequear el cual realiza:
    - Verifica si existe un método main en alguna de las clases.
    - Recorre todas las clases de la TS y verifica para cada una sí:

- Hay herencia circular y existe las clases a la que se extiende.
- Hay un tipo IdClase que no está definido.

- En cada EntradaClase hay método llamado consolidar(), el cual consolida la clase. Su operatoria es la siguiente:

Si 'no está consolidada la clase':

    Si 'el padre no está consolidado'

        consolidar la clase padre.

    consolidarAtributos()
    consolidarMetodo()

Poner como consolidada la clase

ConsolidarAtributo()

Recorrer los atributos del padre

    Si no hay un atributo en el hijo con el mismo nombre

        Agrego atributo de padre al hijo

ConsolidarMetodo()

Recorer los métodos del padre

    Si no hay un método compatible en el hijo

        Agrego el método del padre al hijo

    Sino

Si los tipos de retorno son iguales, los parámetros son del mismo tipo en el mismo orden

Agrego el método del padre al hijo si es compatible

Logros

Se intentan cumplir los logros de:

- Multi-constructor
- Nombres de métodos sobrecargados
- Recuperación errores de declaración