Proyecto Compiladores e Interpretes

Etapa 3

Analizador Semántico

Rodrigo Díaz Figueroa

LU 97400

Clases utilizadas y decisiones de diseño

Cambios desde la última etapa:

Se cambio el nombre de la clase Principal por Minijavac.
Se agregaron las clases SemanticException, ManejadorTS y las correspondientes al árbol de clases de la Tabla de Simbolos, los Tipos y algunas otras.

Clases:

Minijavac
AnalizadorLexico
AnalizadorSintáctico
ManejadorTS
TablaSimbolos
EntradaTS
EClase
EMiembro
EMetodo
EConstructor
EParametro
Eatributo
Tipo
TipoSimple
TipoClase
Visibilidad
FormaMetodo
Bloque
EntradaSalida
Token
Utl
LexicoException
SintacticException
SemanticException

Nuevas clases:

ManejadorTS:
Se encarga de crear la tabla de símbolos y sus entradas, entre ellas las entradas de las clases predefinidas Object y System.
Mantiene a la clase y el miembro actual y realiza algunos chequeos semánticos principalmente ligados a los nombres repetidos.

TablaSimbolos:
Esta es la clase raíz del árbol de la tabla de símbolos. Mantiene un conjunto de entradas de clase y una referencia al método main. Brinda métodos para determinar si una clase esta definida y obtener una clase por su nombre.

EntradaTS:
Clase abstracta de la que heredan todas las clases de entradas de la tabla de símbolos

EClase:
Entrada de Clase de la TS.
Mantiene listas de sus atributos, métodos y constructores. También conoce el nombre y la entrada de clase de su clase ancestro y su método main si es que posee uno.
Brinda métodos para agregar miembros a sus listas, acceder a sus atributos y para realizar su consolidación.

EMiembro:
Clase abstracta de la que heredan las clases EMetodo y EConstructor.
Mantiene una lista de parámetros y una referencia a la entrada de la clase en donde esta definido.

EMetodo:
Entrada de Metodo de la TS.
Tiene un tipo de retorno y una forma de método. Brinda métodos para acceder a sus atributos y para saber si el método es un main. También brinda métodos para comparar una entrada de método con otra, utilizados para saber si dos métodos son iguales para redefinición y si tienen el mismo nombre y aridad.

EConstructor:
Entrada de Constructor de la TS.
Brinda un método para comparar entradas de constructores.

EParametro:
Entrada de Parámetro y Variable de la TS.
Tiene un Tipo y un nombre representado por un Token.
Brinda un método para comparar entradas de parámetros.

EAtributo:
Entrada de Atributo de la TS. Hereda de EParametro.
Tiene ademas una Visibilidad.
Brinda un método para comparar entradas de atributos.

Visibilidad:
Enumerado que representa la visibilidad de un atributo.

FormaMetodo:
Enumerado que representa la forma de un método.

Tipo:
Clase abstracta de la que heredan TipoSimple y TipoClase
Representa los a los Tipos de los atributos, métodos, variables y parámetros.
Define métodos abstractos para comparar dos Tipos, y para chequear que un tipo este definido en la tabla de símbolos.

TipoSimple:
Hereda de Tipo.
Representa a los tipos que no son de tipo clase y al tipo void.
Define los métodos para comparar dos Tipos y chequear que un tipo este definido en la tabla de símbolos

TipoClase:
Hereda de Tipo.
Representa a los tipos de clase.
Define los métodos para comparar dos Tipos y chequear que un tipo este definido en la tabla de símbolos

SemanticException:
Clase que hereda de Exception y es utilizada por distintas clases para lanzar excepciones de tipo semánticas.

Bloque:
Esta clase es utilizada como placeholder para la siguiente etapa y para debugging.

Decisiones de diseño:

- Se optó por mantener a la Tabla de Símbolos como una variable global. Esta se encuentra en la clase Utl como un atributo estático publico

- Se utiliza una clase “handler” para manejar a la Tabla de Símbolos que es ManejadorTS. Esta clase se encarga de crear todas las entradas de la TS manteniendo la clase y el ambiente actual.

- Los chequeos semánticos de nombres duplicados de clases, atributos, métodos y parámetros son realizados al momento de insertarlos en la tabla de símbolos.

- La clase ManejadorTS es la encargada de crear las entradas de las clases predefinidas.

- Se optó por representar a los Tipos con un diagrama de clases utilizando herencia.

- La Tabla de Símbolos mantiene una referencia al método main de una de sus entradas de clases. En el caso de que este definido mas de un método main, se la referencia apuntara al último encontrado según el orden en el que quedo organizado el HashSet de clases que la TablaSimbolos mantiene.

- 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.**
- **Arreglo de todo bien declarado.**
- **Multi-constructor.**
- **Nombres de métodos sobrecargados.**

Logros que se intentan alcanzar de la etapa previa:

- **Arreglo de todo.**
- **Asignación inline.**

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:

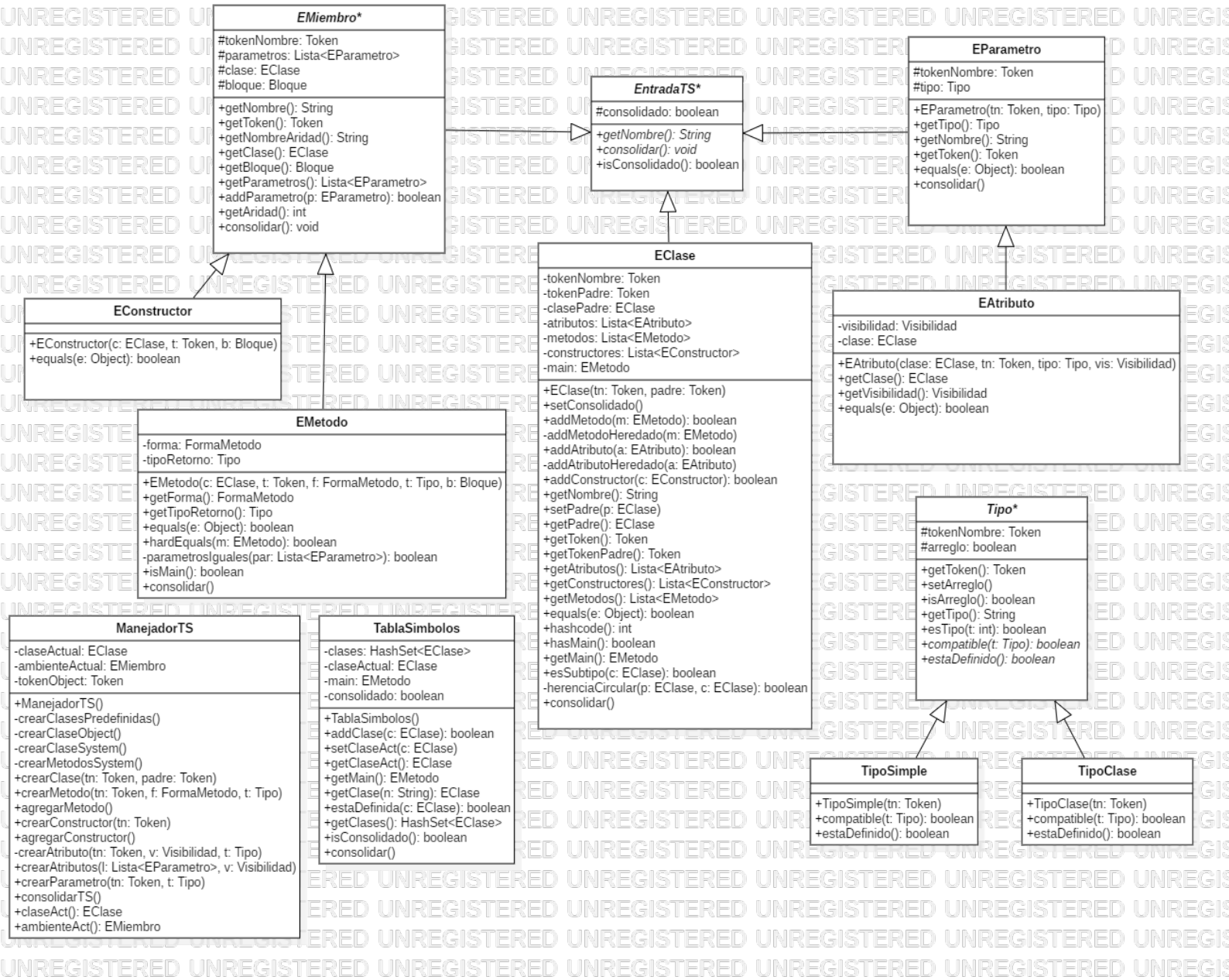
```
>java Minijavac ./Test/Correctos/Test1.java
```

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

Diagrama de clases de la Tabla de Símbolos.



Esquema de traducción y gramática actualizada.

El **AnalizadorSintactico** utiliza a la clase **ManejadorTS** para crear todas las entradas de la tabla de símbolos y luego consolidarla.

{ Antes de inicio se crean las clases predefinidas Object y System con sus respectivos métodos y se agregan a la Tabla de Símbolos a través del ManejadorTS }
Inicio → Clase Clases $
{ Luego de inicio se consolida la Tabla de Símbolos }

Clases → Clase Clases
Clases → ε

Clase → class
{ Se guarda el token con el nombre de la clase }
idClase Herencia
{ Se guarda el token con el nombre del padre de la clase que retorna Herencia, se crea la clase, se agrega a la Tabla de Símbolos y se la setea como clase actual }
{ Miembros }

Herencia → extends
{Se guarda el token de nombre de la clase}
idClase
{Retornar el token guardado}

Herencia → ε
{Retornar un token nulo}

Miembros -> Miembro Miembros
Miembros -> ε
Miembro -> Atributo
Miembro -> Ctor
Miembro -> Metodo

Atributo → Visibilidad
{ Se guarda la visibilidad que fue retornada por Visibilidad }
Tipo
{ Se guarda el tipo. Se crea una lista de variables (EParametro) con el tipo para pasarle como parámetro a ListaDecVars }
ListaDecVars
{ Se crean los atributos usando el método crearAtributos del ManejadorTS con la lista de parámetros completa que ListaDecVars completo y la visibilidad guardada }
Inicializacion ;

Inicializacion -> = Expresion
Inicializacion -> ε

Metodo → **FormaMetodo**
{ Se guarda la forma del método que retorna FormaMetodo }
TipoMetodo
{ Se guarda el tipo de retorno del método que retorna TipoMetodo }
idMetVar
{ Se guarda el token del nombre del método, se crea un método con la forma, tipo de retorno y nombre por medio del ManejadorTS y se lo setea como miembro actual }
ArgsFormales
{ Luego de que ArgsFormales haya insertado todos los parámetros del método, se agrega el método a la tabla de símbolos por medio del método agregarMetodo del ManejadorTS }
Bloque

Ctor → **idClase**
{ Se guarda el token del nombre del constructor, se crea el constructor y se lo setea como miembro actual por medio del ManejadorTS }
ArgsFormales
{ Luego de que ArgsFormales haya insertado todos los parámetros del constructor, se agrega el constructor a la tabla de símbolos por medio del método agregarConstructor del ManejadorTS }
Bloque

ArgsFormales -> (ListaArg)
ListaArg -> Arg ArgFormales
ListaArg -> ε
ArgFormales -> , Arg ArgFormales
ArgFormales -> ε

Arg → **Tipo**
{ Se guarda el tipo del parámetro que retorna Tipo}
idMetVar
{ Se guarda el token del nombre del parámetro, se crea el parámetro y se agrega al miembro actual a través del método crearParametro del ManejadorTS }

FormaMetodo → **static**
{ Retorna el enumerado de FormaMetodo static }

FormaMetodo → **dynamic**
{ Retorna el enumerado de FormaMetodo static }

Visibilidad → **public**
{ Retorna el enumerado de Visibilidad public }

Visibilidad → **private**
{ Retorna el enumerado de Visibilidad private }

TipoMetodo → **Tipo**
{ Retorna el tipo que retorna Tipo }

TipoMetodo → **void**

{ Se guarda el token de nombre y se crea un nuevo objeto de TipoSimple con el token para ser retornado }

Tipo → **boolean**
{ Se guarda el token de nombre y se crea un nuevo objeto de TipoSimple con el token. Se pasa el objecto como parámetro a PosibleArreglo}
PosibleArreglo
{ Se retorna el tipo que retorna PosibleArreglo}

Tipo → **char**
{ Se guarda el token de nombre y se crea un nuevo objeto de TipoSimple con el token. Se pasa el objecto como parámetro a PosibleArreglo}
PosibleArreglo
{ Se retorna el tipo que retorna PosibleArreglo}

Tipo → **int**
{ Se guarda el token de nombre y se crea un nuevo objeto de TipoSimple con el token. Se pasa el objecto como parámetro a PosibleArreglo}
PosibleArreglo
{ Se retorna el tipo que retorna PosibleArreglo}

Tipo → **idClase**
{ Se guarda el token de nombre y se crea un nuevo objeto de TipoClase con el token. Se pasa el objecto como parámetro a PosibleArreglo}
PosibleArreglo
{ Retorna el tipo que retorna PosibleArreglo }

Tipo → **String**
{ Se guarda el token de nombre y se crea un nuevo objeto de TipoSimple con el token. Se pasa el objecto como parámetro a PosibleArreglo}
PosibleArreglo
{ Se retorna el tipo que retorna PosibleArreglo}

{Recibe como parámetro un Tipo}
PosibleArreglo → [ ]
{ Se setea al tipo recibido como parámetro como un arreglo y se retorna}

PosibleArreglo → ***ε***
{ Se retorna el tipo recibido como parámetro }

{Recibe como parámetros un Tipo y una lista de EParametro}
ListaDecVars → **idMetVar**
{Se guarda el token de nombre del parámetro, se crea un EParametro con el token y el tipo y se agrega a la lista}
{Se pasa como parámetro el Tipo y la lista actualizada a ListaDV}
ListaDV
{ Recibe como parámetros un Tipo y una lista de Eparametro }

ListaDV → , idMetVar
{ Se guarda el token de nombre del parámetro, se crea un EParametro con el token y el tipo y se agrega a la lista }
{ Se pasa como parámetro el Tipo y la lista actualizada a ListaDV }
ListaDV

ListaDV → ε

Bloque -> { Sentencias }
Sentencias -> Sentencia Sentencias
Sentencias -> ε
Sentencia -> ;
Sentencia -> if (Expresion) Sentencia SentenciaElse
Sentencia -> while (Expresion) Sentencia
Sentencia -> return Expresiones ;
Sentencia -> Asignacion ;
Sentencia -> SentenciaLlamada ;
Sentencia -> Tipo ListaDecVars Inicializacion ;
Sentencia -> Bloque
SentenciaElse -> else Sentencia
SentenciaElse -> ε
Expresiones -> Expresion
Expresiones -> ε
Asignacion -> AccesoVar = Expresion
Asignacion -> AccesoThis = Expresion
SentenciaLlamada -> (Primario)
Expresion -> ExpOr
ExpOr -> ExpAnd ExpOrR
ExpOrR -> || ExpAnd ExpOrR
ExpOrR -> ε
ExpAnd -> ExpIg ExpAndR
ExpAndR -> && ExpIg ExpAndR
ExpAndR -> ε
ExpIg -> ExpComp ExpIgR
ExpIgR -> OpIgual ExpComp ExpIgR
ExpIgR -> ε
ExpComp -> ExpAd ExpCompR
ExpCompR -> OpComp ExpAd
ExpCompR -> ε
ExpAd -> ExpMul ExpAdR
ExpAdR -> OpAd ExpMul ExpAdR
ExpAdR -> ε
ExpMul -> ExpUn ExpMulR
ExpMulR -> OpMul ExpUn ExpMulR
ExpMulR -> ε
ExpUn -> OpUn ExpUn
ExpUn -> Operando
OpIgual -> ==
OpIgual -> !=
OpComp -> <
OpComp -> >
OpComp -> <=

OpComp -> >=
OpAd -> +
OpAd -> -
OpUn -> +
OpUn -> -
OpUn -> !
OpMul -> *
OpMul -> /
Operando -> Literal
Operando -> Primario
Literal -> null
Literal -> true
Literal -> false
Literal -> intLiteral
Literal -> charLiteral
Literal -> stringLiteral
Primario -> idMetVar MetodoVariable
Primario -> ExpresionParentizada
Primario -> AccesoThis
Primario -> LlamadaMetodoEstatico
Primario -> LlamadaCtor
MetodoVariable -> ArgsActuales Encadenado
MetodoVariable -> Encadenado
ExpresionParentizada -> (Expresion) Encadenado
Encadenado -> . idMetVar Acceso
Encadenado -> AccesoArregloEncadenado
Encadenado -> ε
Acceso -> LlamadaMetodoEncadenado
Acceso -> AccesoVarEncadenado
AccesoThis -> this Encadenado
AccesoVar -> idMetVar Encadenado
LlamadaMetodo -> idMetVar ArgsActuales Encadenado
LlamadaMetodoEstatico -> idClase . LlamadaMetodo
LlamadaCtor -> new LlamadaCtorR
LlamadaCtorR -> idClase LlamadaCtorIDClase
LlamadaCtorR -> TipoArreglo [Expresion] Encadenado
LlamadaCtorIDClase -> ArgsActuales Encadenado
LlamadaCtorIDClase -> [Expresion] Encadenado
TipoArreglo -> char
TipoArreglo -> boolean
TipoArreglo -> int
TipoArreglo -> String
ArgsActuales -> (ListaExpresiones)
ListaExpresiones -> Expresion ListaExp
ListaExpresiones -> ε
ListaExp -> , Expresion ListaExp
ListaExp -> ε
LlamadaMetodoEncadenado -> ArgsActuales Encadenado
AccesoVarEncadenado -> Encadenado
AccesoArregloEncadenado -> [Expresion] Encadenado

Chequeos semánticos realizados y errores detectados.

A continuación se listaran y explicaran brevemente los chequeos semánticos realizados y las estrategias utilizadas para cada uno.

Nombres duplicados:

El chequeo de nombres duplicados es realizado en la primera pasada al momento de insertar un nuevo elemento en los conjuntos. Para esto se redefinió el método equals de la clase Object que es utilizado por las colecciones para comparar los elementos.
Como se optó por realizar el logro de Multi-constructores y Nombres de métodos sobrecargados, se considera que dos miembros (métodos o constructores) son iguales si tienen el mismo nombre y aridad.
La clase ManejadorTS es la encargada de lanzar la excepción al encontrarse con un error de nombre duplicado.
Se detectan todos los errores de nombres duplicados.

Nombre de constructor distinto de nombre de clase:

Este chequeo también es realizado en la primera pasada y es ManejadorTS la clase responsable de hacerlo. Para ello simplemente compara el nombre del constructor con el nombre de la clase actual.

Declaración de tipo clase definida:

Este chequeo se realiza en la segunda pasada al momento de consolidar la Tabla de Símbolos.
La tabla de símbolos le pide a cada una de sus clases que se consolide.
El primer chequeo que una clase realiza para su consolidación es si su clase padre está definida. Para esto se utiliza el método getClase(String nombre) de la clase TablaSimbolos. Si la clase padre está definida, se guarda su referencia, si la clase padre no está definida se reporta el error.
Para las demás declaraciones de tipo clase, tanto en atributos, retornos de métodos o argumentos, el chequeo es delegado a la clase Tipo.
Un TipoSimple siempre está definido, un TipoClase está definido si la clase esta definida. La clase TipoClase utiliza el método estaDefinida(String nombre) de TablaSimbolos para saber si su clase está definida, y en caso de que no lo este, se reporta el error.

Herencia circular:

La herencia circular es chequeada en la consolidación de una clase luego de chequear que su clase padre este definida. Para esto se aprovecha que cada clase tiene una referencia a su clase padre, y se utiliza recursión para recorrer el árbol directo de ancestros hasta llegar a la clase Object o a la clase desde la que se partió.
Si existe herencia circular se reporta el error.

Redefinición de métodos:

El chequeo de la redefinición de métodos se realiza al momento en que se agregan los métodos de una superclase a una subclase cuando se está consolidando una clase.
Para el agregado de métodos heredados se utiliza un método privado (addMetodoHeredado) similar al método de agregar un método normal (addMetodo) pero con la diferencia de que si se encuentra un método con nombre y aridad igual dentro de los métodos de la subclase no se lanza excepción, sino que se pregunta si el método a agregar es exactamente igual, utilizando el método hardEquals de la clase Emetodo, y si no lo es, se lanza excepción normalmente (de nombres duplicados), en el otro caso no se realiza ninguna acción ya que el método está redefinido.

No redefinición de atributos:

El chequeo de la no redefinición de atributos se realiza al momento en que se agregan los atributos de una superclase a una subclase cuando se está consolidando una clase. El chequeo es simplemente antes de agregar el atributo heredado, chequear que no haya nombres duplicados con los atributos de la subclase. En caso de que haya nombres duplicados se reporta el error

Generación de constructor por defecto:

Es chequeo se realiza al momento de consolidar una clase, antes de agregar los miembros heredados de la superclase. El chequeo simplemente consiste en agregar un nuevo constructor sin parámetros si es que el conjunto de constructores de la clase está vacío.

Existencia de método main:

Este chequeo lo realiza la clase TablaSimbolos en la segunda pasada a medida que va recorriendo sus clases para consolidarlas. El chequeo consiste en preguntarle a cada clase si es que tiene un método main, y en el caso de que si lo tenga se lo obtiene y se guarda la referencia. Al final, luego de recorrer todas las clases, se chequea que al menos exista un método main, y en el caso que ninguna clase tenga uno se reporta el error.