Informe Analizador Semántico Parte I

Mauro Montano LU:108882

Modificaciones

Se realizaron las modificaciones de los errores indicados en la corrección de la Etapa 2 y en el código del analizador sintáctico se agregó lo especificado por la EDT. Para esto, se agregaron nuevas clases para el manejo de la Tabla de Símbolos.

Compilar y ejecutar

El programa fue desarrollado en Eclipse y para realizar una compilación desde línea de comandos, es necesario situarse en la carpeta donde tenemos el exportable .jar y ejecutar el siguiente comando:

java –jar Etapa3.jar <Entrada>

Donde Entrada es un parámetro obligatorio, con la ruta del archivo fuente a ser evaluado en el análisis léxico incluyendo el formato de archivo. El resultado se mostrará por pantalla.

Convenciones

Solo puede haber un método estático main sin parámetros en todo el código fuente. De lo contrario muestra el error correspondiente.

Decisiones de diseño

- Se creó un nuevo paquete llamado TablaDeSimbolos donde se encuentran cada una de las clases implementadas de la tabla de símbolos.
- Cuando se llama al método consolidar de la tabla de símbolos, el mismo se esparce por cada una de las clases y de esta manera puede hacer una buena delegación de tareas.
- La delegación de los chequeos no solamente queda en las clases, sino que las clases pueden llamar a métodos provistos por otras clases para hacer algunos chequeos particulares, tales como: el tipo de retorno válido y el chequeo de la signatura en la clase método, o también el chequeo de si es un tipo válido en la clase variable

Funcionamiento del chequeo semántico

- La clase Principal crea el analizador Léxico y el analizador Sintactico, al cual le pasa el analizadorLexico. En la clase Analizador Sintactico se instancia la clase Tabla de Simbolos, de esta manera mientras hacemos el análisis sintáctico vamos agregando los objetos necesarios a la tabla de símbolos y haciendo los chequeos que sean posibles.
- Una vez que termina la pasada del analizador sintáctico se llama al método consolidar de la tabla de símbolos. Este método chequea si existe un solo main estático sin parámetros y luego llama al consolidar de cada clase que tiene guardada en su tabla.
- En el método consolidar de cada clase se realiza lo siguiente: Se fija que no esté actualizada para poder consolidarla. Chequeo que la clase que herede sea una clase existente. Chequeo la herencia circular. Si la clase que hereda no está actualizada, entonces la llamo a consolidar (que también la actualiza).
- Consolido los atributos, para cada atributo de la clase se hace: Si no es un tipo válido entonces damos un error. Chequeo que los atributos de la clase no tengan el mismo nombre que algún atributo de la clase ancestro.
- Consolido los métodos, para cada uno de los métodos se hace: Chequeo que el tipo de retorno sea válido. Chequeo que el tipo de los argumentos sea válido. Chequeo que haya una correcta redefinición de los métodos.
- Chequeo que el tipo de los argumentos del constructor sean válido, para esto verifico si se definió un constructor, si lo hay entonces se hace ese chequeo. De lo contrario, se crea un constructor y se lo agrega a la clase.
- Agrego aquellos métodos de la clase ancestro que no estén redefinidos en esta clase.
- Agrego los atributos de la clase ancestro en esta clase.
- Por último, se setea el flag de que la clase está actualizada.

EDT

<Clase> → **class idClase**{Clase nueva = new Clase(IdClase)
ts.claseActual=nueva ts.insertarClase(nueva)}
<Herencia> {ts.claseActual =
setHerencia(<Herencia>.hereda)}
{ts.claseActual=null}

<Herencia> → **extends idClase** {<Herencia>.hereda = idClase.lexema} | {<Herencia>.hereda = (“Object”) }

<Constructor>→ **idClase**
{Constructor nuevo = new Constructor(idClase)

```
            ts.unidadActual= nuevo
            ts.claseActual.insertarConstructor(nuevo)
          } <ArgsFormales> <Bloque>
          {ts.unidadActual=null}
```

<Atributo>→ <Visibilidad> <Tipo>
{<ListaDecAtrs>.Tipo=<Tipo>.tipo
<ListaDecAtrs>.Visibilidad=<ListaDecAtrs>.Visibilidad
} <ListaDecAtrs> **;**

<Visibilidad>→ **public** {<Visibilidad>.Visibilidad=public} | **private** {<Visibilidad>.Visibilidad=private}

<Tipo> → <TipoPrimitivo> {<Tipo>.tipo= <TipoPrimitivo>.tipo} | **idClase** {<Tipo>.tipo= new TipoClase(idClase)}

<TipoPrimitivo> → **boolean** {<TipoPrimitivo>.tipo= new TipoBoolean()} | **char** {<TipoPrimitivo>.tipo= new TipoChar()} | **int** {<TipoPrimitivo>.tipo= new TipoInt()} | **String** {<TipoPrimitivo>.tipo= new TipoString()}

<ListaDecAtrs>→**idMetVar**

```
{
  if(ts.unidadActual==null){
          Atributo nueva = new Atributo (idMetVar, <ListaDecAtrs>.tipo,
          < ListaDecAtrs>.Visibilidad)
  ts.claseActual.insertarAtributo(nueva)
  < ListaDecAtrsF>.tipo=<ListaDecAtrs>.tipo
 <ListaDecAtrsF>.Visibilidad=<ListaDecAtrsF>.Visibilidad
}<ListaDecAtrsF>
```

<ListaDecAtrsF>→**,** {<ListaDecAtrs>.tipo=<ListaDecAtrsF>.tipo
<ListaDecAtrs>.Visibilidad=<ListaDecAtrsF>.Visibilidad} <ListaDecAtrs> | ε

<Metodo>→<FormaMetodo>

```
 {
        <MetodoF>.Forma = <FormaMetodo>.Forma
 } <MetodoF>
{
   ts.MetodoActual=null
}
```

<FormaMetodo>→**static** {<FormaMetodo>.Forma= static} | **dynamic** {<FormaMetodo>.Forma= dynamic}

<MetodoF>→<TipoMetodo>**idMetVar**

```
{
    Metodo nuevo = new Metodo (idMetVar,
        <MetodoF>.Forma, tipo)
        ts.unidadActual=nuevo ts.clase
        Actual.insertarMetodo(nuevo)
}

 <ArgsFormales><Bloque>
```

<ArgFormal>→ <Tipo> **idMetVar**

```
{
        Variable nueva = new Variable(idMetVar,<Tipo>.tipo)
            ts.unidadActual.insertarArgumento(nueva)
}
```

<Sentencia>→..... | <Tipo> {<ListaDecAtrs>.tipo= <Tipo>.tipo} <ListaDecAtrs> **;** | ...

<TipoMetodo>→ <Tipo> { <TipoMetodo>.tipo=<Tipo>.tipo} | **void** {<TipoMetodo>.tipo= new TipoVoid() }

Errores detectados

Clase existente: Este ocurre cuando se quiere declarar una clase que ya fue declarada.
Atributo repetido en misma clase: Ocurre cuando se busca declarar un atributo que ya fue declarado en la misma clase.
Atributo con mismo nombre que un método: Ocurre cuando un atributo tiene el mismo nombre que un método.
Método repetido: Si se declara un método que tiene el mismo nombre que uno declarado en la misma clase.
Constructor mal nombrado: Si declaramos un constructor que no tiene el mismo nombre que la clase.
Doble constructor: Si existen dos constructores en una misma clase.
Parámetros repetidos: Si tenemos parámetros con el mismo nombre.
Herencia propia: Cuando una clase hereda de sí misma.
No existe un main: Cuando no existe un método main estático sin parámetros en alguna clase.
Hay más de un main: Se da cuando hay más de un main estático sin parámetros en el código fuente.
Clase predefinida: Cuando se quiere declarar una clase con el nombre Object o System.
Clase extendida inexistente: Cuando se quiere extender de una clase que no existe.
Herencia circular: Cuando una clase es ancestra y descendiente de si misma.
Tipo atributo válido: Cuando un atributo no tiene un tipo válido. Esto se puede dar solamente cuando el atributo es de un tipo de clase que no existe.
Tipo de retorno inválido: Cuando un método tiene un tipo de retorno de una clase que no existe.

Tipo argumento inválido: Cuando el tipo de un argumento es de una clase que no existe.
Redefinir un método que tiene los atributos de otro tipo: Se quiere redefinir un método pero los tipos de los atributos son diferentes.
Redefinir un método que tiene diferente cantidad de atributos: Se quiere redefinir un método pero tienen diferente cantidad de atributos.
Redefinir un método que no tiene el mismo modificador de método: Se quiere redefinir un método pero no se tiene el mismo modificador de método.
Redefinir un método que no tiene la misma visibilidad: Se quiere redefinir un método pero no se tiene la misma visibilidad.
Redefinir un método con distinto tipo de retorno: Se quiere redefinir un método que tiene un diferente tipo de retorno.

Casos de prueba

Test validos

- **TipoClase:** Se crean métodos estáticos y dinámicos que tienen el tipo de retorno clase.
- **DelaraciónAtributos:** La clase tiene en distintas líneas la declaración de atributos que son de tipo clase y primitivos.
- **DeclaraciónAtributosMismaLinea:** La clase tiene la declaración de atributos en una misma línea que son de tipo clase y primitivos.
- **RedefiniciónMetodo:** Se redefine un método que está en muchas clases arriba de la subclase.
- **HerenciaSinRedefinición:** Se hereda de una clase sin redefinir métodos.
- **ClasesExtiendenSystemObject:** Una clase que extiende de object y otra clase que extiende de system.
- **MetodoConParámetros:** Un método de la clase tiene parámetros que son de tipo clase y también primitivos.
- **ConstructorConParametros:** El constructor tiene los parámetros de tipo clase y los parámetros de tipo primitivo.
- **SinConstructor:** Falta el constructor de la clase, por lo tanto se crea.
- **AtributoRedefinido:** Se prueba el logro donde existe en una clase el atributo con el mismo nombre que un atributo de la clase padre.

Test Inválidos

- **ArgumentoInvalido:** El tipo del argumento es de una clase que no existe.
- **ExtendidaNoExiste:** La única clase hereda de una clase que no existe.
- **HerenciaCircular:** Hay tres clases que tienen herencia circular.
- **HerenciaPropia:** Se define una clase que hereda de si misma.
- **MainConParametros:** La clase tiene un método estático main que contiene parámetros.
- **TipoInvalido:** El método retorna un tipo que no existe.
- **TipoAtributoInválido:** La clase posee un atributo que tiene un tipo de clase que no existe.
- **DobleMain:** Existen dos mains estáticos sin parámetros.
- **AtributoNombreMetodo:** Un atributo con el mismo nombre que un método es declarado

antes del método.

- **ClaseExistente:** Hay dos clases con el mismo nombre.
- **ConstructorMalNombrado:** El constructor no tiene el nombre de la clase.
- **ParametrosRepetidos:** En un constructor hay dos parámetros repetidos.
- **ClaseObject:** Se define una clase con el nombre Object.
- **ClaseSystem:** Se define una clase con el nombre System.
- **DobleConstructor:** Existen dos constructores en la clase
- **FaltaMain:** Falta la clase main.

Logros obtenidos

- **Entrega anticipada**

- **Atributos redefinidos:** Se permite que en una clase se declaren atributos con el mismo nombre que los definidos en sus superclases.

  Ejemplo de un programa valido con la redefinición del atributo elem:

```
class A{
        public int elem;
}

class B extends A{
        private char x;
}

class C extends B{
        private String elem;

        static void main(){}
}
```