

**DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA****ASUNTO: Solicitud de Actividades**

Celaya, Gto., 2018-05-07

LENGUAJES Y AUTÓMATAS II

DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ

ACTIVIDAD 4 (VALOR 70 PUNTOS)

LEA CUIDADOSAMENTE, Y REALICE LAS SIGUIENTES ACTIVIDADES, CONSIDERANDO LOS CRITERIOS DE CALIDAD PROPUESTOS EN LOS DOCUMENTOS DE LA GUÍA TUTORIAL, Y LA RÚBRICA DE EVALUACIÓN.

1. TOMANDO EN CUENTA QUE LA SEGUNDA ETAPA DEL ANÁLIZADOR SINTÁCTICO SE HA CONCLUIDO SATISFACTORIAMENTE Y CON ELLO EL EQUIPO ACREDITÓ LA EVALUACIÓN ANTERIOR, AHORA EL SIGUIENTE PASO SERÁ QUE EN EQUIPO INVESTIGUEN, DISEÑEN E IMPLEMENTEN **LA TERCERA FASE O ETAPA DEL PROYECTO**, ES DECIR UN ANALIZADOR SEMÁNTICO.

MUY IMPORTANTE:

SI LA EVALUACIÓN ANTERIOR NO FUÉ ACREDITADA, ESO SIGNIFICA QUE DEBERÁ PRIMERO IMPLEMENTAR DESDE EL INICIO TODOS LOS PLANTEAMIENTOS NECESARIOS PARA TENER UN ANÁLISIS SINTÁCTICO LIBRE DE ERRORES. DE LO CONTRARIO, SI PERSISTEN LAS FALLAS DETECTADAS EN LA EVALUACIÓN ANTERIOR, Y NO SE CORRIGEN, ESTA ETAPA TAMBIÉN PRESENTARÁ FALLOS.

2. LA ACTIVIDAD COMO EQUIPO DEBERÁ COMENZAR CON LA INVESTIGACIÓN Y FUNDAMENTACIÓN DE LOS SIGUIENTES TEMAS.

- INVESTIGAR.** ¿ QUÉ ES UN ANÁLISIS SEMÁNTICO APLICADO A LA VALORACIÓN DE UN LENGUAJE ?
- INVERTIGAR.** ¿ EN QUÉ CONSISTE UN ANÁLISIS SEMÁNTICO Y QUÉ LO CARACTERIZA ?
- IDENTIFICAR.** ¿ QUÉ CASOS DE ESTUDIOS SON LOS IMPORTANTES A CONSIDERAR EN EL ANÁLISIS SEMÁNTICO ?
- INVESTIGAR Y PROPONER.** ¿ QUÉ PROCESOS Y PROBLEMAS ATIENDE UN ANÁLISIS SEMÁNTICO ?
- INVESTIGAR.** ¿ CÓMO IMPLEMENTAR UN ANÁLISIS SEMÁNTICO ?

NOTA : ESTOS TEMAS DEBEN SER EL RESULTADO DE UNA INVESTIGACIÓN Y ANÁLISIS COMO EQUIPO, Y NO UNA TRANSCRIPCIÓN DE TEMAS AISLADOS, PUES EN TODO MOMENTO LAS FUENTES DE CONSULTA DEBEN SER LA BASE DEL DESARROLLO DE ESTE PUNTO Y SUS APARTADOS.





3. COMO EQUIPO Y DERIVADO DEL ANÁLISIS ANTERIOR SE DEBEN PROPONER LOS ALGORITMOS Y ESTRUCTURAS DE DATOS NECESARIAS PARA LA IMPLEMENTACIÓN DE UN PROTOTIPO DE ANALIZADOR SEMÁNTICO. ES DECIR, CREACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS PARA DETERMINAR SI TIENEN EL SENTIDO CORRECTO LOS ELEMENTOS INTEGRADOS COMO CÓDIGO FUENTE.

CONCRETAMENTE EN ESTE PUNTO DEBERÁN COMO EQUIPO, REDACTAR Y DETALLAR TODOS LOS ELEMENTOS QUE SE USARÁN PARA LA IMPLEMENTACIÓN DEL ANALIZADOR SEMÁNTICO.

4. PARA EL INCISO C DEL PUNTO 2 ANTERIOR, SE DEBERÁN CREAR PROGRAMAS DE MÍNIMO 25 LÍNEAS (SIN CONSIDERAR LOS COMENTARIOS) CADA UNO, EN LOS CUALES SE CODIFIQUE EN EL LENGUAJE PROTOTIPO, INSTRUCCIONES CON LÓGICA QUE EJEMPLIFIQUEN LOS ERRORES QUE EN CADA CASO DE ESTUDIO SE PROPONGAN.

TALES PROGRAMAS DEBEN ESTAR PERFECTAMENTE DOCUMENTADOS Y CO-
RELACIONADOS CON LOS CASOS DE ESTUDIO CORRESPONDIENTES.

5. CARACTERÍSTICAS QUE LA ACTIVIDAD 4 DEBE POSEER PARA CONSIDERARSE COMPLETA.

- A. UN ANÁLISIS LÉXICO EFICIENTE, LIBRE DE ERRORES Y SOLVENTADAS TODAS LAS OBSERVACIONES REALIZADAS POR EL PROFESOR DURANTE LA EVALUACIÓN PRESENCIAL.
- B. UN ANÁLISIS SEMÁNTICO EFICIENTE, LIBRE DE ERRORES Y SOLVENTADAS TODAS LAS OBSERVACIONES REALIZADAS POR EL PROFESOR DURANTE LA EVALUACIÓN PRESENCIAL.
- C. UNA TABLA DE SÍMBOLOS PERFECTAMENTE ESTRUCTURADA CON LA TOKENIZACIÓN APROPIADA Y SOLVENTADAS TODAS LAS OBSERVACIONES REALIZADAS POR EL PROFESOR DURANTE LA EVALUACIÓN PRESENCIAL.
- D. PLANTEAMIENTO Y FUNDAMENTACIÓN DE LOS CASOS DE USO DEL ANALIZADOR SINTÁCTICO, ASÍ COMO LOS ERRORES EN QUE DERIVARÁ CADA UNO DE ELLOS.
- E. PLANTEAMIENTO Y FUNDAMENTACIÓN DE LOS CASOS DE USO DEL ANALIZADOR SEMÁNTICO, ASÍ COMO LOS ERRORES EN QUE DERIVARÁ CADA UNO DE ELLOS.
- F. PREPARACIÓN DE LOS PROGRAMAS SUFICIENTES, ESCRITOS EN EL LENGUAJE PROTOTIPO, QUE APOYEN LA COMPROBACIÓN DE CADA CASO DE ESTUDIO. TALES PROGRAMAS DEBERÁN ESTAR COMPLETAMENTE DOCUMENTADOS.
- G. GENERACIÓN DE UN CATÁLOGO DE ERRORES, CORRELACIONADO A LOS CASOS DE USO PROPUESTOS.
- H. DISEÑO DE UNA SOLUCIÓN MODELADA EN OBJETOS, APOYADA EN DIAGRAMAS DE CLASE QUE DESCRIBAN LA ARQUITECTURA PROPUESTA PARA EL PROTOTIPO DEL ANALIZADOR SEMÁNTICO.





I. MODELADO DE UNA INTERFACE GRÁFICA CON LAS CARACTERÍSTICAS INDICADAS EN CLASE Y EN LA EVALUACIÓN PRESENCIAL PASADA.

J. MODELADO E IMPLEMENTACIÓN DE UN PROCESO DE :

GENERACIÓN DE UNA INTERFACE GRÁFICA DE USUARIO => LECTURA DE LA TABLA DE SÍMBOLOS => GENERACIÓN DE ÁRBOLES DE DERIVACIÓN SINTÁCTICA => REVISIÓN DE LOS TOKENS, SEGÚN LA SINTÁXIS PROPUESTA EN EL LENGUAJE PROTOTIPO => VALIDACIÓN SINTÁCTICA DE LOS CASOS DE ESTUDIOS IDENTIFICADOS Y PROPUESTOS => VALIDACIÓN SEMÁNTICA DE LOS CASOS DE ESTUDIOS IDENTIFICADOS Y PROPUESTOS.

K. GENERACIÓN DE UN CALENDARIO DE ACTIVIDADES PLANIFICADAS VS. ACTIVIDADES REALIZADAS.

L. GENERACIÓN DE UNA BITÁCORA DE INCIDENCIAS.

M. TODAS LAS EVIDENCIAS GENERADAS Y REUNIDAS DEBERÁN INTEGRARSE AL UN ARCHIVO PDF, NOMBRADO COMO SE INDICA MÁS ADELANTE.

ESTAS EVIDENCIAS PODRÁN ELABORARSE CON HERRAMIENTAS ELECTRÓNICAS, COMO PROCESADOR DE TEXTO, DE IMÁGENES, HOJAS DE CÁLCULO, ETC.

N. EL NÚCLEO DE CADA ALGORITMO CODIFICADO TAMBIÉN DEBERÁ FORMAR PARTE DEL ARCHIVO DE EVIDENCIAS.

6. COMO EQUIPO GENERAR SU PORTAFOLIO DE EVIDENCIAS ELECTRÓNICAS (PED).

ESTO SE HARÁ USANDO UN CD O DVD EN EL CUAL EL EQUIPO CREARÁ UNA CARPETA PARA INTEGRAR TODAS SUS EVIDENCIAS DE TRABAJO SEMESTRAL, SEGÚN LO INDICA LA ESTRUCTURA PROPUESTA EN LA GUÍA TUTORIAL.

LA ESTRUCTURA RAÍZ DEL CD O DVD PUEDE TENER LA SIGUIENTE APARIENCIA:

```
\  
+-- LAII_EVALUACIÓN_1_SEM_ENE_JUN_2018  
|  
+-- LAII_EVALUACIÓN_2_SEM_ENE_JUN_2018  
|  
+-- LAII_EVALUACIÓN_3_SEM_ENE_JUN_2018  
|  
+-- LAII_EVALUACIÓN_4_SEM_ENE_JUN_2018
```

POR ÚLTIMO, HACER ÉNFASIS EN QUE LA FECHA DE ENTREGA DEL PED, SERÁ EN LA SESIÓN DE EXAMEN PARA LA EVALUACIÓN 4.





NOTAS GENERAL DE LA ACTIVIDAD :

- **LAS ACTIVIDADES INCOMPLETAS NO TENDRÁN VALOR ALGUNO, POR LO TANTO EN EQUIPO SE DEBE REVISAR QUE EL CONTENIDO DE LA ACTIVIDAD ESTÉ COMPLETO.**
- **SE DEBE CONSIDERAR Y TOMAR EN CUENTA PARA EL CORRECTO CUMPLIMIENTO DE ESTA ACTIVIDAD, LO SOLICITADO EN LA GUÍA TUTORIAL, CONCRETAMENTE EN EL **PUNTO 3 INCISO i (Trabajo en equipo).****
- **LAS PRÁCTICAS DEBERÁN CONTENER TODAS LAS SECCIONES SOLICITADAS EN LA GUÍA TUTORIAL, CONCRETAMENTE EL **PUNTO 3 INCISO h (Prácticas).****
- **SE DEBE CONSIDERAR Y TOMAR EN CUENTA PARA EL CORRECTO CUMPLIMIENTO DE ESTA ACTIVIDAD LO SOLICITADO EN LA GUÍA TUTORIAL, CONCRETAMENTE EN EL **PUNTO 6, (Evidencias tipo a).****



**OBSERVACIONES:**

- ✓ LA REVISIÓN SERÁ EN DIVERSAS VERTIENTES. PUEDE SER AL MOMENTO DE SOLICITAR LA CARPETA DE EVIDENCIAS, O BIEN PUEDE SER AL SOLICITAR LA EXPOSICIÓN DE LA TAREA EN LA CLASE. TAMBIÉN PUEDE SER POR SOLICITUD EXPRESA DEL INTERESADO A PARTICIPAR EN CLASE EXponiendo BREVEMENTE SU ACTIVIDAD.
- ✓ AQUELLAS ACTIVIDADES EN FORMATO DIGITAL SE DEBERÁN TENER SIEMPRE, Y EN TODO MOMENTO A LA MANO EN UNA MEMORIA USB.
- ✓ ÉSTAS ACTIVIDADES PODRÁN SER SOLICITADAS EN LA CLASE, O BIEN PARA SU ENVÍO A UNA CUENTA DE CORREO.
- ✓ ESTAS ACTIVIDADES DEBEN ESTAR LISTAS E INTEGRADAS A LA CARPETA DE EVIDENCIAS (FÍSICAMENTE) A LA FECHA DE ENTREGA INDICADA AL FINAL DE ÉSTE DOCUMENTO.
- ✓ CADA HOJA QUE ENTREGUE DE SU ACTIVIDAD, DEBERÁ ESTAR FIRMADA AL MARGEN DERECHO.
- ✓ UNA VEZ ELABORADA SU ACTIVIDAD, RECUERDE DIGITALIZARLA Y NOMBRARLA EN BASE A LA SIGUIENTE NOMENCLATURA.
- ✓ SI SUS EVIDENCIAS ENVIADAS POR CORREO, NO CUMPLEN CON LA NOMENCLATURA SOLICITADA, NO SERÁN CONSIDERADAS COMO EVIDENCIAS PARA SU EVALUACIÓN.
- ✓ CON ESTA ACTIVIDAD, USTED DEBERÁ IR INTEGRANDO SUS CARPETAS FÍSICA Y ELECTRÓNICA DE EVIDENCIAS, Y AL FINAL DEL SEMESTRE EN UN DISCO COMPACTO HARÁ ENTREGA DE SU CARPETA ELECTRÓNICA DE EVIDENCIAS.
- ✓ PARA TENER DERECHO A LA REVISIÓN Y EVALUACIÓN DE SUS ACTIVIDADES, DEBE REGISTRAR SU ASISTENCIA A CLASE, EL DÍA SEÑALADO PARA LA ENTREGA DE LA MISMA.
- ✓ FALTAR A CLASE EL DÍA DE LA ENTREGA, ANULA LA REVISIÓN DE SUS EVIDENCIAS.
- ✓ POR ÚLTIMO, POR FAVOR GESTIONE APROPIADAMENTE SU TIEMPO, Y SEA PUNTUAL EN SU ENTREGA.
- ✓ AÚN PARA TRABAJOS EN EQUIPO APPLICAN TODAS LAS MISMAS OBSERVACIONES ANTERIORES.



**LA NOMENCLATURA SOLICITADA ES :**

AAAA-MM-DD_MATERIA_DOCUMENTO_EQUIPO_NOCTROL_APELLIDOS_NOMBRE_SEM.PDF

(NOTA : * TODO EN MAYÚSCULA ***)****DONDE :**

AAAA : **AÑO**
 MM : **MES**
 DD : **DIA**
 MATERIA : **SO, TSO, LAII, LI**
 DOCUMENTO : **A1-ACTIVIDAD 1, P1-PRACTICA 1, R1-REPORTE 1, T1-TAREA 1, PG1-PROGRAMA,
ETC. (CAMBIANDO EL NÚMERO CONSECUATIVO POR EL QUE CORRESPONDA)**
 EQUIPO : **NÚMERO DEL EQUIPO QUE CORRESPONDA SEGÚN INDICACIÓN DEL PROFESOR.**
 NOCTROL : **SU NÚMERO DE CONTROL**
 APELLIDOS : **SUS APELLIDOS**
 NOMBRE : **SU NOMBRE**
 SEM : **EL PERÍODO SEMESTRAL EN CURSO: ENE-JUN / AGO-DIC**

EJEMPLO :

2018-05-21_LAII_A4_EQUIPO_99_9999999_PEREZ_PEREZ_JUAN_ENE-JUN18.PDF

FECHA DE ENTREGA:

**VÍA CORREO ELECTRÓNICO, EL LUNES 21 DE MAYO DEL 2018, CON HORA LÍMITE DE ENTREGA
HASTA LAS 14:00 HORAS (2 DE LA TARDE).**

**DESPUÉS DE ESTA HORA, LA ACTIVIDAD SERÁ CONSIDERADA COMO EXTEMPORÁNEA Y NO
CONTARÁ COMO EVIDENCIA PARA SU EVALUACIÓN.**

MUY IMPORTANTE:

**POR FAVOR ANEXE A SU ARCHIVO .PDF DE EVIDENCIAS, ESTA SOLICITUD DE ACTIVIDADES CON
TODAS LAS HOJAS FIRMADAS EN EL MARGEN DERECHO.**

CALENDARIO 2017-2018 - EVALUACIÓN 4

MAYO						
L	M	M	J	V	S	D
1	2	3	4	5	6	
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUNIO						
L	M	M	J	V	S	D
			1	2	3	
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

EVALUACIÓN FORMATIVA DE 1^{er}OPORTUNIDAD (PARCIALES)

2A - El análisis semántico

La semántica estudia el significado, sentido o interpretación de los signos lingüísticos (símbolos, palabras o expresiones).

Análisis semántico aplicado a la valoración de un lenguaje.

El análisis semántico hace una verificación de tipos, en donde verifica si los operandos de cada operador son compatibles con base en nuestro lenguaje propuesto, o que los parámetros actuales de una función sean coherentes respecto a los parámetros formales de esta. Además de verificar que la sucesión de tokens utilizados dentro del código fuente sea correcta semánticamente.

2B - ¿En qué consiste un análisis semántico y que lo caracteriza?

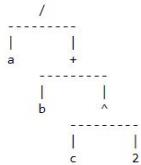
Esta es la tercera fase del compilador, en esta se hace uso del árbol sintáctico previamente generado en la fase anterior (análisis sintáctico), ya que con el uso de este y la tabla de símbolos se buscarán errores semánticos.

Como se mencionó anteriormente la verificación de tipos buscará que el operador tenga operandos cuyos tipos coincidan. En esta fase del análisis, se prepara la generación del código.

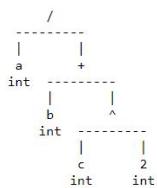
Sea la expresión

```
int a,b,c;
a/(b+c^2)
```

El árbol sintáctico es:



De la instrucción declarativa, la tabla de símbolos y el analizador morfológico obtenemos los atributos de los operandos:



En los compiladores de un solo paso, la función del análisis se ejecuta junto con el analizador sintáctico.

En los compiladores de una o más pasadas, el análisis se ejecuta independiente de la generación del código, haciendo uso de un archivo intermedio, el cual contiene información del árbol sintáctico de forma linealizada. Esto para facilitar su manejo y hacer posible su almacenamiento en memoria auxiliar.

2C Casos de estudio

Validación de tipos

Que las variables utilizadas dentro de un código fuente no coincidan con el tipo de operador utilizado.

Ejemplo:

- Supongamos que se declara la siguiente variable tipo entero
 - int x = 2;
- Si después se le quiere sustituir el valor de x por una cadena, este sería semánticamente incorrecto.
 - x = "Hola Mundo";

El analizador deberá enviar error cuando la validación de tipos sea incorrecta, el error producido será el 320.

Pase de parámetros incorrectos en una función

Se refiere a este caso cuando los pases de parámetros no coinciden con la definición formal de la función ya sea porque la cantidad de parámetros es incorrecta o porque los tipos de datos no coinciden con los parámetros formales.

Ejemplo:

- Se declara una función con los siguientes parámetros.

- func CheckThisOut(string g){
 - Si se hace uso de la función y el parámetro es erróneo es un error semántico.
 - CheckThisOut(int x);
 - Si se hace uso de la función y la cantidad de parámetros es incorrecta.
 - CheckThisOut(int x, string y);
 - CheckThisOut();

El error generado por el analizador en este caso es el 330.

Falta de declaración de variable o función

Si no se declara una variable, y esta es usada en instrucciones posteriores.

Ejemplo:

- Si no se declara la variable x, y se utiliza para hacer una asignación de valor .
 - $x = 2$
 - Generaría error ya que la variable utilizada no ha sido declarada.

El analizador deberá generar un error cuando este caso se presente, el error será el 310.

Declaración de una variable o función con un nombre ya asignado

Si se declara una variable o función con un nombre ya utilizado anteriormente esto generará el error 340.

Ejemplo:

Retorno de dato incorrecto

Este error se genera en los casos de que el return dentro de una función no coincide

```
func res (double num1, int num2 ) : int {  
    return "resta";  
}
```

Este caso genera el error 350.

Función sin retorno

El error se presenta cuando una función que retorna algún tipo de dato no contiene la sentencia return en su declaración.

Ejemplo:

```
func suma (int num1, int num2 ) : int {  
    int res = num1 + num2;  
} // Esta función es incorrecta  
// ya que se esperaba un return
```

El error generado por este caso es el error 360.

Sentencia inesperada

Es cuando se encuentra una sentencia irrelevante o fuera de lugar en el código. Por ejemplo, si existe un return fuera de una función, esto nos generará el error 370.

```
#####
# CASO DE ESTUDIO 1 y 2 #
# Validación de tipos #
# Pase de parámetros incorrectos de una función #
#####

func sum (int snum1 , int snum2) : int {
    real ssuma;
    ssuma = snum1 + snum2;           //Suma es un tipo de dato decimal y no se le puede
                                    //asignar un tipo de dato entero, por lo que
                                    //generará un error.
    return ssuma;
}

func res (real rnum1, int rnum2 ) : int {
    int rresta;
    rresta = rnum1 - rnum2;         //El error sera por que se hace la resta
                                    //de un tipo de dato decimal con un entero
    return "resta";                //El tipo de retorno no coincide
}

if(suma>resta){
    print "La suma es mayor que la resta";
}else{
    suma = "ERROR" - 1;           //La operación que se está realizando es
                                    //entre dos tipos de dato diferentes.
}
res("hola", 4.4);               //Los parámetros actuales son diferente a los
                                    //parámetros formales por lo que generará un
                                    //error

sum(10, 20, 30);               //El pase de parámetros es incorrecto ya que
                                    //son más de lo que la función requiere, genera
                                    //error

if(suma>resta) {

    print "La suma es mayor que la resta";
    print "Esto es error";
} else {
    print "La resta es mayor que la suma";
    print "Esto es un error";
}

int num1 = "uno";                //La asignación a la variable num1 de tipo de
                                    //dato entero es una cadena.
real num3 = 15.02;
string num4 = 1642;              //La asignación es entre dos tipos de datos
                                    //distintos este error
int suma = sum (num1 , num2);
int resta = res (num1 , num2);
print "la suma es";
```

```
print suma;
print "la resta es";
print resta;

#####
#           CASO DE ESTUDIO 3 y 4
#           Falta de declaración de variables y funciones
# Declaración de una variable o función con un nombre ya declarado #
#####

func fact (int fanum): int {
    if (fanum == 1) {
        return 1;
    } else {
        return fanum;
    }
}

func fact (int finum): int {      //la función fact ya fue creada con anterioridad
    //por lo tanto es considerado como error.
    if (finum == 1) {
        return 1;
    } else if (finum == 2) {
        return 1;
    } else {
        return finum;
    }
}

int num = 0;
int fibo;

real fibo;                      //fibo es una variable previamente definida como
int facto;                      //tipo entero, así que este es un error.

println "Dame un numero para hacer unos calculos:";
print ">";
read Cheese;                     //Variable Cheese no ha sido creada, por lo tanto
facto = fact(num);               //es un error semántico.

fibo = meow(num);                //La función meow no ha sido creada
                                //lo cual es considerado como un error sintáctico

println "El factorial de ";
print Cheese;
print " es: ";
print facto;
println "El numero de fibonacci de ";
print Cheese;
print " es: ";
print fibo;
```

```
#####
#           CASO DE ESTUDIO 5, 6 y 7          #
#           Retorno de dato incorrecto          #
#           Función sin retorno               #
#           Sentencia inesperada             #
#####

func saludar ( ) : string {           //La estructura de esta función
    string hola = "hola";           //es la correcta
    string mundo = "mundo";
    string saludo;
    saludo = hola + mundo;
    return saludo ;
}
func multiplicar ( real num1 , real num2 ) : int {      //La función espera de
    real mul ;           //retorno un tipo de dato
    mul = num1 * num2 ;
    return mul ;           //int y no real y esto
                           // genera un error.
}
func despedida ( ) : string {           //La función espera un return
    string h = "hasta";           //de tipo string y este no existe, por lo
    string l = "luego";           // que genera un error
    string des;
    des = h + l;

}

func div ( real num3 , real num4 ) : real {
    real divi ;
    divi = num3 * num4 ;
    return divi ;
}

int x = 7;
int y = 5;

if (x > 5){
    println "X es mayor que Y";      //Esto es una sentencia inesperada, ya que
    return x;                      // dentro del if no se esperaba un return
}

string texto = saludar();
println texto;
texto = despedida();
println texto;
```

2D Procesos y problemas que atiende el analizador semántico

El analizador semántico cumple principalmente la función de identificar que los tipos de datos sean declarados congruentemente así como que todas las variables están definidas previamente a ser usadas.

Para validar que los tipos de datos coinciden en una operación se realiza un proceso de propagación de atributos, el cual puede ser top-down o bottom-up, dependiendo si la propagación va de arriba hacia abajo en el árbol o de abajo hacia arriba respectivamente.

El otro proceso que sigue es el de recorrer los árboles sintácticos y la tabla de símbolos para dar respuesta a si un identificador ya ha sido declarado anteriormente, así como sus parámetros formales para que al sustituirlos correspondan con los que se les asignan.

2E Implementación del analizador semántico

Para la implementación del analizador semántico se usaron las estructuras de datos que se generaron en el analizador sintáctico y en el analizador léxico, más específicamente, la tabla de símbolos y los árboles sintácticos.

Para realizar el análisis de la presente etapa, se generaron tablas hash para guardar las funciones e identificadores que van siendo declaradas en el código. De este modo, es posible identificar que no haya repeticiones y que los identificadores hayan sido declarados cuando se quieran usar.

Para realizar el proceso de verificación de los tipos de datos, el cual es una de las funciones esenciales del analizador, se toman en cuenta dos casos. Cuando es una expresión, ya sea aritmética o relacional, se valida que los datos incluidos en la expresión sean del mismo tipo. Cuando se trata de una asignación a una variable se hace referencia a la tabla de símbolos para validar que el tipo del dato, retorno o expresión siendo asignado a la variable es igual al tipo de la variable en cuestión.

También, se crea una tabla de variables temporal cuando se inicia la declaración de una función para manejar el contexto. Así, cuando se usa una variable o un return se valida el contexto para saber cómo se va a tratar. En el caso del return, si no hay contexto, este no debería estar declarado y genera un error semántico. Esta tabla es destruida al salir del contexto para optimizar memoria.

REFERENCIAS:

- <http://www.lcc.uma.es/~galvez/ftp/libros/Compiladores.pdf>
- <http://arantxa.ii.uam.es/~alfonsec/docs/compila5.htm>

Errores del 1 al 99 “reservados para el sistema”

Todo error relacionado a procesos del sistema, como errores de permisos, de lectura de archivos, y otros.

- **Error 10.** Error de archivo inexistente.
 - Este error se presenta cuando un archivo al que se hace referencia no existe.
- **Error 11.** Error de lectura de archivo.
 - Se presenta cuando ocurre un error de entrada/salida al leer cualquier archivo.
- **Error 12.** Programa incompleto
 - Este error se presenta cuando el programa llega a su fin de manera prematura.

Errores del 100 al 199 “léxicos”

Todo error relacionado a la etapa del análisis léxico entra en esta categoría. Son pocos, pero hay espacio para expansión futura.

- **Error 110.** Error de token inesperado
 - Se presenta cuando el analizador no puede reconocer el token después de compararlo con todos los casos posibles.
- **Error 120.** Error uso de símbolos no definidos en el alfabeto.
 - El error se presenta si en cualquier parte del código fuente es detectado un símbolo que no pertenece al lenguaje.

Errores del 200 al 299 “sintácticos”

Todo error relacionado a la etapa del análisis sintáctico entra en esta categoría.

- **Error 210.** Falta delimitador ";" al final de sentencia.
 - Este error se presenta cuando al final de una secuencia establecida en el lenguaje no se usa el delimitador ";".
- **Error 220.** Return solo acepta un parámetro seguido de un punto y coma.
 - Este error se presenta cuando después del return se intentan hacer operaciones complejas después de la palabra reservada.
- **Error 230.** Apertura y cierre de llaves y corchetes errónea.
 - Este error se genera cuando las llaves o corchetes aparecen sin su par. Es decir, que si se abren no se cierran o que aparezca uno de cierre sin haberse abierto uno.
- **Error 240.** Mala estructuración de sentencias según el lenguaje definido.
 - Este error es dado cuando la estructura de una sentencia no sigue la estructura definida por el lenguaje, ya sea que se agreguen elementos o falten.
- **Error 250.** Estructura no reconocida por el lenguaje.
 - Este error es generado cuando hay una sucesión de tokens válidos pero que directamente no siguen ni se asemejan a ninguna estructura definida.

Errores del 300 al 399 “semánticos”

- **Error 310.** La variable o función no ha sido declarada.
 - Este error se presenta cuando se hace uso de una variable o una función que no ha sido declarada anteriormente.
- **Error 320.** Validación de tipos incorrecta.
 - Este error se presenta cuando se le asigna un tipo que no le corresponde a una variable, cuando se realizan operaciones entre diferentes tipos de datos.
- **Error 330.** Pase de parametros incorrecto
 - Este error se genera cuando el pase de parámetros no coincide con el de la declaración de la función; ya sea por tipo o por nombre.
- **Error 340.** Asignación de nombre de variable o función ya asignado.
 - El error se presenta cuando se encuentra en el programa fuente una declaración de variable o función que ya ha sido declarada anteriormente.
- **Error 350.** Retorno de dato incorrecto
 - El error se presenta cuando el retorno de una función no corresponde con el tipo de dato que debe regresar.
- **Error 360.** Función sin retorno
 - El error se presenta cuando una función que retorna algún tipo de dato no contiene la sentencia return en su declaración.
- **Error 370.** Sentencia inesperada
 - El error se presenta cuando se encuentra una sentencia irrelevante o fuera de lugar en el código.

Reglas semánticas

Estructuras a las que se le aplicarán las reglas semánticas. Tomar en cuenta que para cualquier estructura que contenga identificadores, se validará que los mismos hayan sido declarados con anterioridad. Además, si se declara un nuevo identificador, se verificará que no haya sido declarado previamente.

- **Declaración de funciones**

- Si hay tipo de dato de retorno, verificar que exista el retorno y corresponda al tipo de dato declarado.
- Para los parámetros que sean usados dentro de la función, tomar en cuenta el contexto.

- **Invocación de funciones**

- Comparar el número de parámetros y el tipo de estos con los registrados en la tabla de funciones, para validar que correspondan.

- **Expresiones aritméticas, relacionales.**

- Verificar que el tipo de los operandos coincidan.

- **Ciclo for**

- Verificar que el renglón contenga solo tipos de datos entero.
- Verificar que el contador sea entero.

- **Asignación de valores**

- La asignación debe ser congruente con el tipo de dato declarado.

- **Estructuras sintácticas restantes**

- Verificar lo establecido previamente con los identificadores
- Verificar que estructuras anidadas cumplan con las reglas previamente establecidas (bloques, expresiones, ciclos, etc.)

Hashtable para variables

Para llevar un control de las variables que han sido declaradas, se optó por usar una hashtable, de modo que sea fácil verificar si el uso de las mismas es válido.

- **Key:** Nombre o lexema
- **Value:** ID único en la tabla de símbolos para obtener información

De igual modo, la clase tendrá un campo del tipo String llamado **contexto**, para identificar las tablas temporales usadas dentro de una función.

Hashtable para funciones

Para llevar un control de las funciones que han sido declaradas, se usará una hashtable.

- **Key:** Nombre o lexema
- **Value:** Un registro descrito a continuación

CAMPO	DESCRIPCIÓN
String ID	ID único en la tabla de símbolos
int T_RETURN	Tipo de dato que la función retorna
ArrayList<Integer> T_PARAMS	Lista de los tipos de dato de los parámetros que acepta la función (si existen) en orden

```
<bloque> ::= "{" <sentencias> "}"  
  
<sentencias> ::= <sentencia> <sentencia>*  
<sentencia> ::= <declaración> | <asignación> | <llamada método> | <ciclos>  
| <condicionales> | <lectura> | <impresión> | <return>  
  
<llamada a método> ::= <id> "(" "[<parámetros>]"")"  
<método> ::= func <id> "(" "[<argumentos>]"")" [: <tipo>] <bloque>  
<parámetros> ::= <id> | <número> | <decimal> | <cadena> | <parámetros> ,  
<id> | <parámetros> , <número> | <parámetros> , <decimal> | <parámetros> ,  
<cadena>  
<argumentos> ::= <tipo> <id> | <argumentos>, <tipo> <id>  
<retorno> ::= return ; | return <id> ; | return <número> ; | return  
<decimal> ; | return <cadena> ; | return <booleano> ;  
<ciclos> ::= <ciclo for> | <ciclo while>  
<condicionales> ::= <if then else> | <if then>  
  
<if then else> ::= <if then> else <bloque> | <if then> else <if then else>  
<if then> ::= if "(" "<expresión relacional>"")" <bloque>  
  
<ciclo for> ::= for "(" <id> in <rango> ")" <bloque>  
<ciclo while> ::= while "(" "<expresión relacional>"")" <bloque>  
<rango> ::= <número> -> <número> | <número> -> <id> | <id> -> <número> |  
<id> -> <id>  
  
<lectura> ::= read <id> ;  
<impresión> ::= (print|println) <id>; | (print|println) <cadena> ;  
  
<asignación> ::= [int] <id> = <número> ; | [real] <id> = <decimal>; |  
[string] <id> = <cadena> ; | [bool] <id> = <booleano> ; | [bool] <id> =  
<expresión relacional>; | [<tipo>] <id> = <expresión aritmética>; |  
[<tipo>] <id> = <llamada a método>; | [<tipo>] <id> = <id>; | <tipo> <id>;  
<expresión aritmética> ::= (<id>|<número>|<decimal>|<cadena>) <operador  
aritmético> (<id>|<número>|<decimal>|<cadena>)  
<expresión relacional> ::= (<id>|<número>|<decimal>|<cadena>) <operador  
relacional> (<id>|<número>|<decimal>|<cadena>)
```

```
<operador relacional> ::= < | > | <= | >= | == | !=  
<operador aritmético> ::= + | - | * | /  
<operador> ::= ++ | -- | ->
```

```
<id> ::= (<letra>|_|$)(<letra>|<dígito>|_|$)*
```

```
<decimal> ::= <número>."<número>
```

```
<número> ::= <dígito><dígito>*
```

```
<cadena> ::= \"<símbolo>*\\"
```

```
<booleano> ::= true | false
```

```
<letra> ::= <letra minúscula> | <letra mayúscula>
```

```
<letra minúscula> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n  
| ñ | o | p | q | r | s | t | u | v | w | x | y | z
```

```
<letra mayúscula> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N  
| Ñ | O | P | Q | R | S | T | U | V | W | X | Y | Z
```

```
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

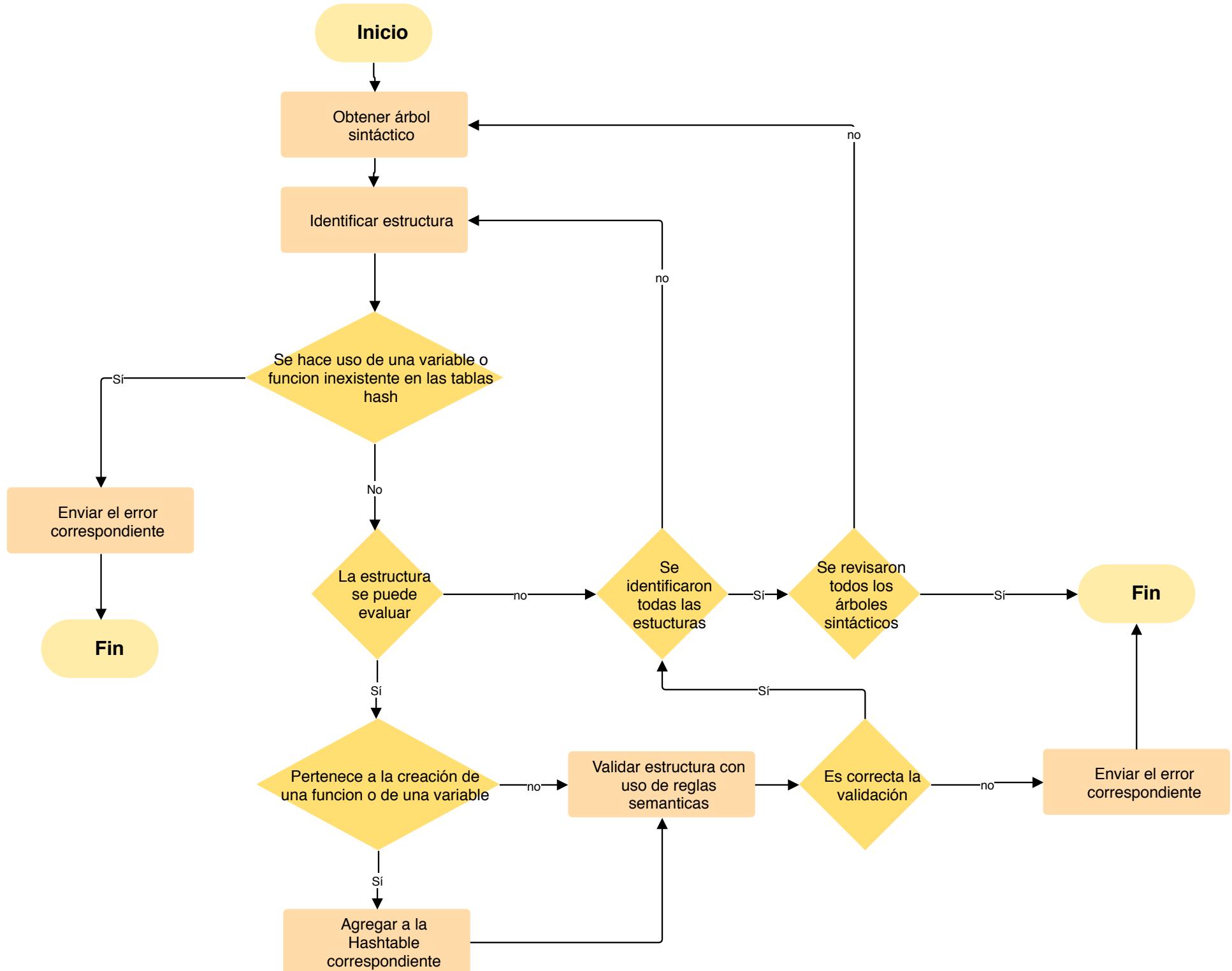
```
<delimitadores> ::= ; | " " | \t
```

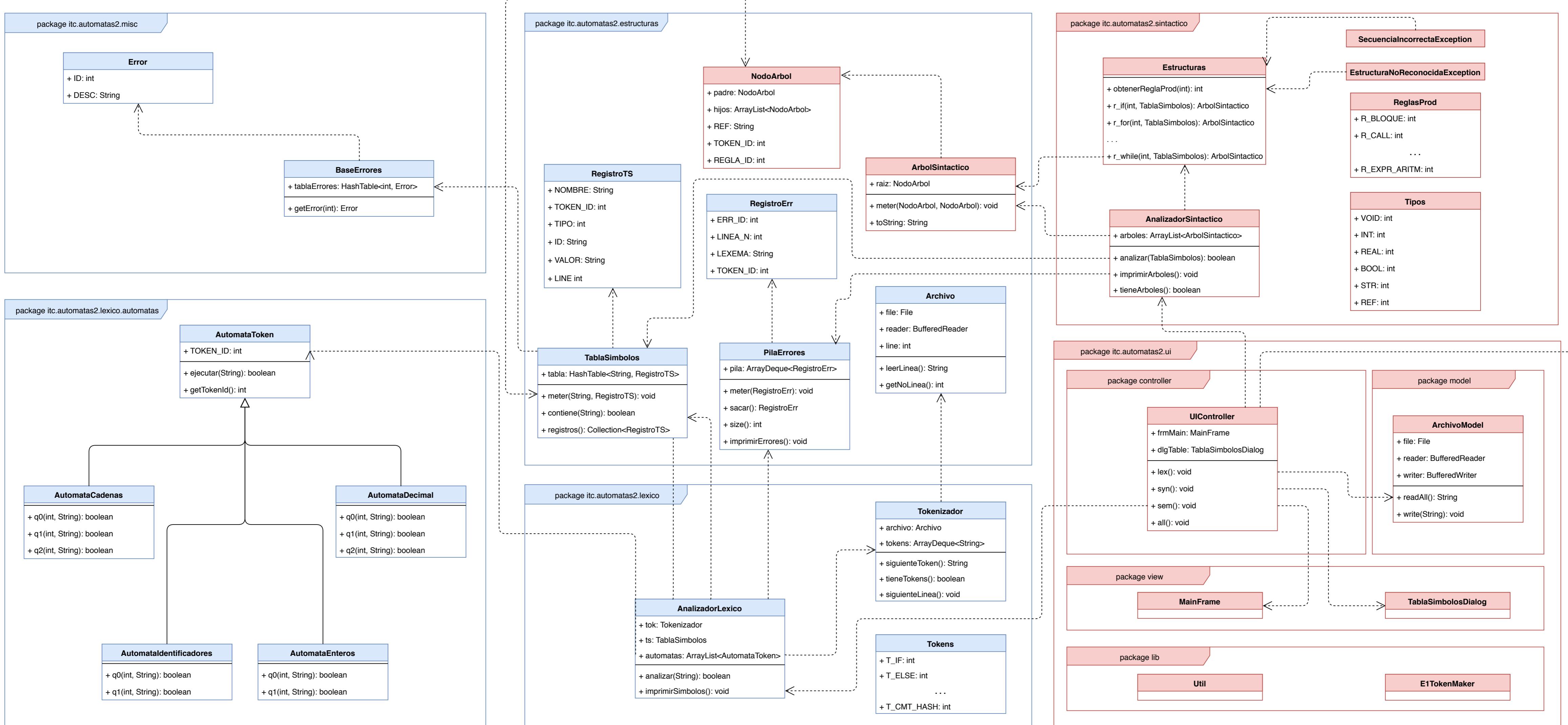
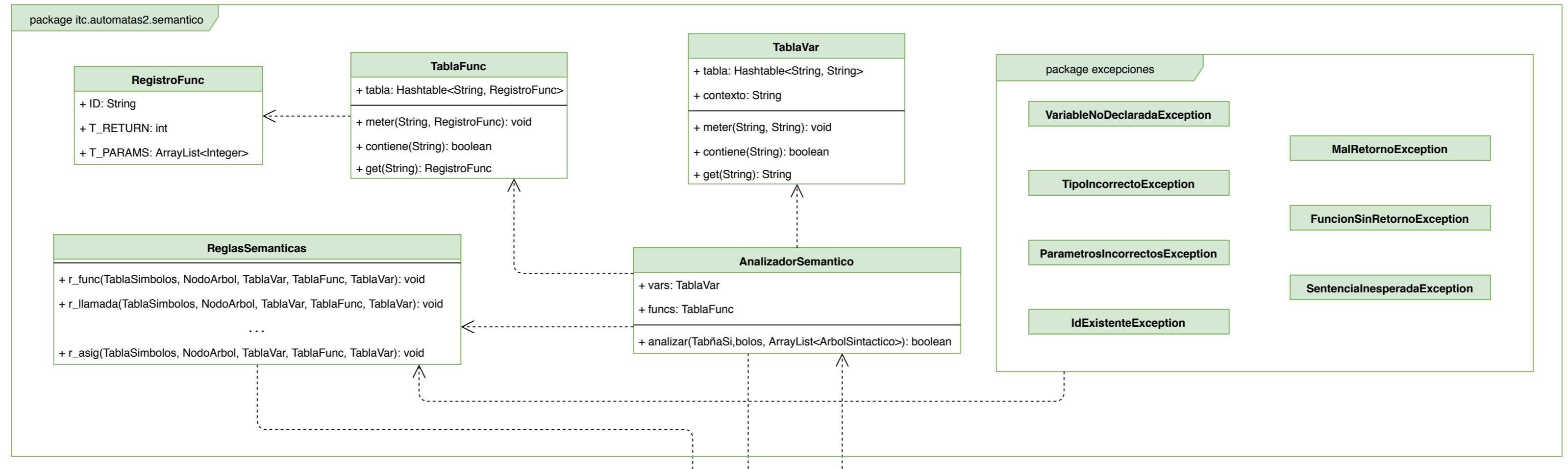
```
<comentario> ::= //<símbolo>*\n | #<símbolo>*\n
```

```
<tipo> ::= int | real | bool | string
```

```
<reservadas> ::= if | else | while | for | in | func | return | true |  
false | null | int | real | bool | string | print | println | read
```

```
<símbolos> ::= <caracteres ascii imprimibles>
```





EQUIPO 1		ACTIVIDAD 4																		LENGUAJES Y AUTÓMATAS II				
ACTIVIDADES / FASES		TIEMPOS POR ACTIVIDAD																		TOTAL POR ACTIVIDAD		OBSERVACIONES		
		05/05	06/05	07/05	08/05	09/05	10/05	11/05	12/05	13/05	14/05	15/05	16/05	17/05	18/05	19/05	20/05	21/05	PLANEADO	REAL				
Correcciones de la actividad 3																								
Correcciones al diseño		60																		60	30	Correcciones menores, ver bitácora de incidencias		
Correcciones al código		60																		60	150	Ver bitácora de incidencias		
Actividades iniciales																								
Lectura inicial de la solicitud de actividad			60																	60	60			
Investigación del marco teórico y discusión acerca de los temas				120	120															240	240			
Análisis																								
Planteamiento del problema				120																120	120			
Documentación del marco teórico				120	120															240	240			
Diseño																								
Definición de casos de estudio						120														120	120			
Categorización de errores						60														60	75	Ver bitácora de incidencias		
Diagramas de flujo						60														240	240			
Diagramas de clases						240														240	240			
Construcción																								
Implementación de componentes							480	480	480	360										1,800	1,620			
Integración de componentes							120	480	480	480	60									600	540			
Actividades finales																								
Recopilación de la documentación (sólo responsable del equipo)																			60	60	60	60		
Presentación de la actividad (sólo responsable del equipo)																			15	15	15	15		
														minutos				3915		3750				
														horas/hombre				16.31		15.63				

1. Como sugerencia del profesor en la última revisión presencial, se realizó una corrección menor a la definición del lenguaje y al analizador sintáctico, permitiendo declaración de variables sin asignar valor inicial. Esta tarea se realizó antes de que se emitiera la solicitud de la presente actividad.

05 de mayo del 2018

2. Se encontraron errores menores en casos muy especiales del análisis sintáctico (específicamente con las reservadas true y false), los cuales se arreglaron.

17 de mayo del 2018

3. Se encontraron aún más errores en el análisis sintáctico de acuerdo a la especificación del lenguaje y en casos especiales (invocación de métodos), los cuales se corrigieron.
4. Se agregaron un par de errores que no se habían considerado, y se modificaron los casos de estudio.

19 de mayo del 2018

Anexo

Código generado
durante esta etapa

```
1 package itc.automatas2.semantico;
2
3 import java.util.Hashtable;
4
5 /**
6  * Tabla donde se llevara la relacion de las variables definidas en el codigo fuente.
7 */
8 public class TablaVar {
9     private Hashtable<String, String> tabla = new Hashtable<>();
10    public String contexto;
11
12    public TablaVar() {
13
14    }
15
16    /**
17     * Metodo para establecer el contexto de una variable si esta pertenece a alguna funcion
18     *
19     * @param contexto es la funcion a la que pertenece dicha variable si es que pertenece
20     */
21    public TablaVar(String contexto) {
22        this.contexto = contexto;
23    }
24
25    /**
26     * Metodo para obtener un registro especifico en la tabla de variables
27     *
28     * @param nombre es el nombre de la variable
29     * @return regresara la variable con el nombre especificado
30     */
31    public String get(String nombre) {
32        return tabla.get(nombre);
33    }
34
35    /**
```

```
36     * Metodo para insertar un nuevo registro a la tabla de variables
37     *
38     * @param nombre es el nombre de la variable
39     * @param id      es el ID unico de la variable
40     */
41    public void meter(String nombre, String id) {
42        tabla.put(nombre, id);
43    }
44
45 /**
46     * Metodo donde se validara la existencia de una variable dentro de la tabla de simbolos
47     *
48     * @param nombre es el nombre de la variable
49     * @return regresa un valor booleano que indicara si la variable es existente dentro de la tabla
de variables
50     */
51    public boolean contiene(String nombre) {
52        return tabla.containsKey(nombre);
53    }
54 }
55
```

```
1 package itc.automatas2.semantico;
2
3 import java.util.Hashtable;
4
5 /**
6  * Tabla donde se llevara el registro de las funciones ya establecidas en el codigo fuente.
7 */
8 public class TablaFunc {
9     private Hashtable<String, RegistroFunc> tabla = new Hashtable<>();
10
11    /**
12     * Metodo para obtener un registro especifico en la tabla de variables
13     *
14     * @param nombre es el nombre de la funcion
15     * @return regresa la funcion con el nombre especificado
16     */
17    public RegistroFunc get(String nombre) {
18        return tabla.get(nombre);
19    }
20
21    /**
22     * Metodo para insertar un nuevo registro a la tabla de funcion
23     *
24     * @param nombre es el nombre de la funcion
25     * @param rf      es el registro de la funcion que se generara
26     */
27    public void meter(String nombre, RegistroFunc rf) {
28        tabla.put(nombre, rf);
29    }
30
31    /**
32     * Metodo donde se validara la existencia de una funcion dentro de la tabla de simbolos
33     *
34     * @param nombre es el nombre de la funcion
35     * @return regresa un valor booleano que indica si dicha funcion existe dentro de TablaFunc
```

```
36     */
37     public boolean contiene(String nombre) {
38         return tabla.containsKey(nombre);
39     }
40 }
```

```
1 package itc.automatas2.semantico;
2
3 import java.util.ArrayList;
4
5 /**
6  * Esta clase representara el registro de las funciones que seran almacenadas en la tabla de
7  * funciones (TablaFunc)
8 */
9 public class RegistroFunc {
10    public final String ID;
11    public final int T_RETURN;
12    public final ArrayList<Integer> T_PARAMS;
13
14    /**
15     * Este sera el constructor de nuestros registros de funcion
16     *
17     * @params id
18     * @params t_return
19     */
20    public RegistroFunc(String id, int t_return) {
21        this.ID = id;
22        this.T_RETURN = t_return;
23        this.T_PARAMS = new ArrayList<>();
24    }
25}
```

```
1 package itc.automatas2.semantico;
2
3 import itc.automatas2.estructuras.*;
4 import itc.automatas2.lexico.Tipos;
5 import itc.automatas2.lexico.Tokens;
6 import itc.automatas2.semantico.excepciones.*;
7 import itc.automatas2.sintactico.ReglasProd;
8
9 import java.util.ArrayList;
10
11 /**
12 * Esta clase se encargara de reconocer la estructura a la que se hace uso en el codigo fuente
13 */
14 class ReglasSemanticas {
15     private static boolean returnFound;
16
17     /**
18      * Método encargado de analizar semánticamente la estructura de la declaración de una función.
19      *
20      * @param ts Tabla de símbolos
21      * @param na Nodo del árbol
22      * @param tv Tabla de nombres de variables
23      * @param tf Tabla de nombres de funciones
24      * @param ctx Función a la que pertenece dicha variable
25      * @throws IdExistenteException
26      * @throws VariableNoDeclaradaException
27      * @throws TipoIncorrectoException
28      * @throws ParametrosIncorrectosException
29      * @throws SentenciaInesperadaException
30      * @throws FuncionSinRetornoException
31      * @throws MalRetornoException
32      */
33     static void r_funcTablaSimbolos(NodoArbol na, TablaVar tv, TablaFunc tf, TablaVar ctx)
34         throws IdExistenteException, VariableNoDeclaradaException, TipoIncorrectoException,
35         ParametrosIncorrectosException, SentenciaInesperadaException, FuncionSinRetornoException,
```

```

33 MalRetornoException {
34     RegistroFunc reg = null;
35     for (NodoArbol h : na.hijos) {
36         if (h.TOKEN_ID == Tokens.T_FUN) {
37             String nombre = ts.getByID(h.REF).NOMBRE;
38             if (tf.contiene(nombre)) {
39                 throw new IdExistenteException(
40                     String.format("La función %s ya existe (línea %d).", nombre, ts.getByID(
41                         na.hijos.get(0).REF).LINE)
42                     );
43             } else {
44                 reg = new RegistroFunc(h.REF, getTipo(ts, h.REF));
45                 tf.meter(nombre, reg);
46             }
47         } else if (h.REGLA_ID == ReglasProd.R_ARGS) {
48             for (NodoArbol arg : h.hijos) {
49                 if (arg.TOKEN_ID == Tokens.T_VAR) {
50                     String nombre = ts.getByID(arg.REF).NOMBRE;
51                     ctx.meter(nombre, arg.REF);
52                     reg.T_PARAMS.add(getTipo(ts, arg.REF));
53                 }
54             }
55         } else if (h.REGLA_ID == ReglasProd.R_BLOQUE) {
56             r_bloque(ts, h, tv, tf, ctx);
57         }
58     }
59
60 /**
61 * Método encargado de analizar semánticamente la estructura de la asignación de valores y
62 * la existencia de estas.
63 *
64 * @param ts Tabla de símbolos
65 * @param na Nodo del árbol
66 * @param tv Tabla de nombres de variables

```

```

67     * @param tf Tabla de nombres de funciones
68     * @param ctx Función a la que pertenece dicha variable
69     * @throws IdExistenteException
70     * @throws VariableNoDeclaradaException
71     * @throws TipoIncorrectoException
72     * @throws ParametrosIncorrectosException
73     */
74     static void r_asigTablaSimbolos (TablaSimbolos ts, NodoArbol na, TablaVar tv, TablaFunc tf, TablaVar ctx)
75     throws IdExistenteException, VariableNoDeclaradaException, TipoIncorrectoException,
76     ParametrosIncorrectosException {
77         boolean declarando = esReservadaTipo(na.hijos.get(0).TOKEN_ID);
78         boolean asignando = false;
79         int tipoVar = getTipo(ts, na.hijos.get(0).REF);
80         int tipoAsig = 0;
81
82         for (NodoArbol h : na.hijos) {
83             if (h.REGLA_ID == ReglasProd.R_EXPR_ARITM || h.REGLA_ID == ReglasProd.R_EXPR_REL) {
84                 tipoAsig = r_exp(ts, h, tv, tf, ctx);
85             } else if (h.REGLA_ID == ReglasProd.R_CALL) {
86                 tipoAsig = r_llamada(ts, h, tv, tf, ctx);
87             } else {
88                 switch (h.TOKEN_ID) {
89                     case Tokens.T_VAR:
90                         String nombre = ts.getByID(h.REF).NOMBRE;
91                         if (declarando) {
92                             if (variableDeclarada(nombre, tv, ctx))
93                                 throw new IdExistenteException(
94                                     String.format("La variable \"%s\" ya existe (línea %d).",
95                                     nombre, ts.getByID(h.REF).LINE));
96                         }
97                         if (ctx != null)
98                             ctx.meter(nombre, h.REF);
99                         else
100                             tv.meter(nombre, h.REF);
101                     } else if (!variableDeclarada(nombre, tv, ctx))
102                         throw new VariableNoDeclaradaException(
103                             String.format("La variable \"%s\" no ha sido declarada (línea %d).",
104                             nombre, ts.getByID(h.REF).LINE));
105                 }
106             }
107         }
108     }

```

```

100         throw new VariableNoDeclaradaException(
101             String.format("La variable \"%s\" no existe (línea %d).",
102                         nombre, ts.getByID(h.REF).LINE)
103         );
104     break;
105     case Tokens.T_ASSIGN:
106         asignando = true;
107         break;
108     case Tokens.T_INT_CONST:
109     case Tokens.T_REAL_CONST:
110     case Tokens.T_TRUE:
111     case Tokens.T_FALSE:
112     case Tokens.T_STR_CONST:
113         tipoAsig = getTipo(ts, h.REF);
114         break;
115     }
116 }
117 }
118
119 if (asignando && tipoVar != tipoAsig)
120     throw new TipoIncorrectoException(
121         String.format("No se puede asignar un valor del tipo %s a una variable del tipo
122 %s (línea %d)",
123             Tipos.nombres[tipoAsig], Tipos.nombres[tipoVar], ts.getByID(na.hijos.get
124 (0).REF).LINE)
125         );
126 }
127 /**
128 * Método encargado de analizar semánticamente la estructura de la llamada de un método.
129 *
130 * @param ts Tabla de símbolos
131 * @param na Nodo del árbol
132 * @param tv Tabla de nombres de variables
133 * @param tf Tabla de nombres de funciones

```

```

133     * @param ctx Función a la que pertenece dicha variable
134     * @return
135     * @throws ParametrosIncorrectosException
136     */
137     static int r_llamadaTablaSimbolos (TablaSimbolos ts, NodoArbol na, TablaVar tv, TablaFunc tf, TablaVar ctx)
138     throws ParametrosIncorrectosException {
139         ArrayList<Integer> params = new ArrayList<>();
140         RegistroFunc reg = tf.get(ts.getByID(na.hijos.get(0).REF).NOMBRE);
141         for (NodoArbol h : na.hijos) {
142             if (h.REGLA_ID == ReglasProd.R_PARAMS) {
143                 for (NodoArbol param : h.hijos) {
144                     if (param.TOKEN_ID != Tokens.T_COMMA) {
145                         params.add(getTipo(ts, param.REF));
146                     }
147                 }
148             }
149             if (params.size() != reg.T_PARAMS.size())
150                 throw new ParametrosIncorrectosException(
151                     String.format("El número de parámetros en la llamada a la función \"%s\" no
coincide con su declaración (línea %d).",
152                     ts.getByID(na.hijos.get(0).REF).NOMBRE, ts.getByID(na.hijos.get(0).REF).
LINE)
153                 );
154             for (int i = 0; i < params.size(); i++) {
155                 if (!params.get(i).equals(reg.T_PARAMS.get(i)))
156                     throw new ParametrosIncorrectosException(
157                         String.format("El tipo del parámetro %d en la llamada a la función \"%s\""
no coincide con el de su declaración (línea %d).",
158                         i + 1, ts.getByID(na.hijos.get(0).REF).NOMBRE, ts.getByID(na.hijos.
get(0).REF).LINE)
159                     );
160             }
161         return getTipo(ts, na.hijos.get(0).REF);
162     }

```

```
163
164     /**
165      * Método encargado de analizar semánticamente la estructura de un ciclo for
166      *
167      * @param ts Tabla de símbolos
168      * @param na Nodo del árbol
169      * @param tv Tabla de nombres de variables
170      * @param tf Tabla de nombres de funciones
171      * @param ctx Función a la que pertenece dicha variable
172      * @throws VariableNoDeclaradaException
173      * @throws TipoIncorrectoException
174      * @throws IdExistenteException
175      * @throws ParametrosIncorrectosException
176      * @throws SentenciaInesperadaException
177      * @throws FuncionSinRetornoException
178      * @throws MalRetornoException
179      */
180     static void r_forTablaSimbolos (TablaSimbolos ts, NodoArbol na, TablaVar tv, TablaFunc tf, TablaVar ctx)
181         throws VariableNoDeclaradaException, TipoIncorrectoException, IdExistenteException,
182             ParametrosIncorrectosException, SentenciaInesperadaException, FuncionSinRetornoException,
183             MalRetornoException {
184         NodoArbol actual = na.hijos.get(2);
185         String nombre = ts.getByID(actual.REF).NOMBRE;
186         if (!variableDeclarada(nombre, tv, ctx)) {
187             throw new VariableNoDeclaradaException(
188                 String.format("La variable \"%s\" no existe (línea %d).",
189                             nombre, ts.getByID(na.hijos.get(0).REF).LINE)
190             );
191         }
192         if (getTipo(ts, na.hijos.get(2).REF) != Tipos.INT) {
193             throw new TipoIncorrectoException(
194                 String.format("El contador en un ciclo for debe ser de tipo INT (línea %d).",
195                             ts.getByID(na.hijos.get(0).REF).LINE)
196             );
197     }
```

```

195         int tipoL = getTipo(ts, na.hijos.get(4).REF);
196         int tipoR = getTipo(ts, na.hijos.get(6).REF);
197         if (tipoL != Tipos.INT || tipoR != Tipos.INT)
198             throw new TipoIncorrectoException(
199                 String.format("Los límites del rango en un ciclo for deben ser de tipo INT (
200                     línea %d),",
201                         ts.getByID(na.hijos.get(0).REF).LINE)
202                     );
203         r_bloque(ts, na.hijos.get(8), tv, tf, ctx);
204     }
205
206     /**
207      * Método encargado de analizar semánticamente la estructura de un ciclo while.
208      *
209      * @param ts Tabla de símbolos
210      * @param na Nodo del árbol
211      * @param tv Tabla de nombres de variables
212      * @param tf Tabla de nombres de funciones
213      * @param ctx Función a la que pertenece dicha variable
214      * @throws TipoIncorrectoException
215      * @throws IdExistenteException
216      * @throws VariableNoDeclaradaException
217      * @throws ParametrosIncorrectosException
218      * @throws SentenciaInesperadaException
219      * @throws FuncionSinRetornoException
220      * @throws MalRetornoException
221      */
222     static void r_whileTablaSimbolos (TablaSimbolos ts, NodoArbol na, TablaVar tv, TablaFunc tf, TablaVar ctx)
223         throws TipoIncorrectoException, IdExistenteException, VariableNoDeclaradaException,
224         ParametrosIncorrectosException, SentenciaInesperadaException, FuncionSinRetornoException,
225         MalRetornoException {
226         for (NodoArbol h : na.hijos) {
227             switch (na.REGLA_ID) {
228                 case ReglasProd.R_EXPR_REL:
229                     r_exp(ts, h, tv, tf, ctx);

```

```

226                     break;
227             case ReglasProd.R_BLOQUE:
228                 r_bloque(ts, h, tv, tf, ctx);
229                 break;
230         }
231     }
232 }
233
234 /**
235  * Método encargado de analizar semánticamente la estructura de la instrucción de leer o
236  * imprimir.
237  *
238  * @param ts Tabla de símbolos
239  * @param na Nodo del árbol
240  * @param tv Tabla de nombres de variables
241  * @param tf Tabla de nombres de funciones
242  * @param ctx Función a la que pertenece dicha variable
243  * @throws VariableNoDeclaradaException
244  */
245 static void r_leer_impTablaSimbolos (TablaSimbolos ts, NodoArbol na, TablaVar tv, TablaFunc tf, TablaVar ctx)
246 throws VariableNoDeclaradaException {
247     for (NodoArbol h : na.hijos) {
248         if (h.TOKEN_ID == Tokens.T_VAR) {
249             String nombre = ts.getByID(h.REF).NOMBRE;
250             if (!variableDeclarada(nombre, tv, ctx)) {
251                 throw new VariableNoDeclaradaException(
252                     String.format("La variable \"%s\" no existe (línea %d).",
253                                     nombre, ts.getByID(h.REF).LINE)
254                 );
255             }
256         }
257     }
258 /**

```

```

259     * Método encargado de analizar semánticamente la estructura de un if.
260     *
261     * @param ts Tabla de símbolos
262     * @param na Nodo del árbol
263     * @param tv Tabla de nombres de variables
264     * @param tf Tabla de nombres de funciones
265     * @param ctx Función a la que pertenece dicha variable
266     * @throws TipoIncorrectoException
267     * @throws IdExistenteException
268     * @throws VariableNoDeclaradaException
269     * @throws ParametrosIncorrectosException
270     * @throws SentenciaInesperadaException
271     * @throws FuncionSinRetornoException
272     * @throws MalRetornoException
273     */
274     static void r_ifTablaSimbolos (TablaSimbolos ts, NodoArbol na, TablaVar tv, TablaFunc tf, TablaVar ctx)
275     throws TipoIncorrectoException, IdExistenteException, VariableNoDeclaradaException,
276     ParametrosIncorrectosException, SentenciaInesperadaException, FuncionSinRetornoException,
277     MalRetornoException {
278         for (NodoArbol h : na.hijos) {
279             switch (h.REGLA_ID) {
280                 case ReglasProd.R_EXPR_REL:
281                     r_exp(ts, h, tv, tf, ctx);
282                     break;
283                 case ReglasProd.R_IF:
284                     r_if(ts, h, tv, tf, ctx);
285                     break;
286             }
287         }
288     }
289
290     /**

```

```

291     * Método encargado de analizar semánticamente la estructura de una expresión.
292     *
293     * @param ts Tabla de símbolos
294     * @param na Nodo del árbol
295     * @param tv Tabla de nombres de variables
296     * @param tf Tabla de nombres de funciones
297     * @param ctx Función a la que pertenece dicha variable
298     * @return
299     * @throws TipoIncorrectoException
300     */
301     private static int r_expTablaSimbolos (TablaSimbolos ts, NodoArbol na, TablaVar tv, TablaFunc tf, TablaVar
ctx) throws TipoIncorrectoException, VariableNoDeclaradaException {
302         NodoArbol l = na.hijos.get(0);
303         NodoArbol r = na.hijos.get(2);
304
305         String nombreL = ts.getByID(l.REF).NOMBRE;
306         String nombreR = ts.getByID(r.REF).NOMBRE;
307         if (l.TOKEN_ID == Tokens.T_VAR && !variableDeclarada(nombreL, tv, ctx))
308             throw new VariableNoDeclaradaException(
309                 String.format("La variable \"%s\" no existe (línea %d).",
310                               nombreL, ts.getByID(l.REF).LINE)
311             );
312         if (r.TOKEN_ID == Tokens.T_VAR && !variableDeclarada(nombreR, tv, ctx))
313             throw new VariableNoDeclaradaException(
314                 String.format("La variable \"%s\" no existe (línea %d).",
315                               nombreR, ts.getByID(r.REF).LINE)
316             );
317
318         int tipoL = getTipo(ts, l.REF);
319         int tipoR = getTipo(ts, r.REF);
320         if (tipoL != tipoR)
321             throw new TipoIncorrectoException(
322                 String.format("Los operandos en una expresión aritmética o relacional deben ser
del mismo tipo (línea %d).",
323                               ts.getByID(l.REF).LINE)

```

```

324         );
325         return tipoL;
326     }
327
328     /**
329      * Método encargado de analizar semánticamente la estructura de la declaración de un retorno.
330      *
331      * @param ts Tabla de símbolos
332      * @param na Nodo del árbol
333      * @param tv Tabla de nombres de variables
334      * @param tf Tabla de nombres de funciones
335      * @param ctx Función a la que pertenece dicha variable
336      * @throws TipoIncorrectoException
337      * @throws MalRetornoException
338      */
339     private static void r_retorno(TablaSimbolos ts, NodoArbol na, TablaVar tv, TablaFunc tf,
340     TablaVar ctx) throws TipoIncorrectoException, MalRetornoException {
341         int tipoFunc = tf.get(ctx.contexto).T_RETURN;
342         int tipoRet = getTipo(ts, na.hijos.get(1).REF);
343         if (tipoFunc != tipoRet)
344             throw new MalRetornoException(
345                 String.format("No se puede retornar un tipo %s en una función del tipo %s (%s",
346                 linea %d),
347                 Tipos.nombres[tipoRet], Tipos.nombres[tipoFunc], ts.getByID(na.hijos.get
348                 (0).REF).LINE));
349     }
350
351     /**
352      * Método encargado de analizar semánticamente la estructura del contenido de un bloque.
353      *
354      * @param ts Tabla de símbolos
355      * @param na Nodo del árbol
356      * @param tv Tabla de nombres de variables
357      * @param tf Tabla de nombres de funciones

```

```

356     * @param ctx Función a la que pertenece dicha variable
357     * @throws TipoIncorrectoException
358     * @throws VariableNoDeclaradaException
359     * @throws IdExistenteException
360     * @throws ParametrosIncorrectosException
361     * @throws SentenciaInesperadaException
362     * @throws FuncionSinRetornoException
363     * @throws MalRetornoException
364     */
365     private static void r_bloqueTablaSimbolos (TablaSimbolos ts, NodoArbol na, TablaVar tv, TablaFunc tf,
366     TablaVar ctx) throws TipoIncorrectoException, VariableNoDeclaradaException, IdExistenteException,
367     ParametrosIncorrectosException, SentenciaInesperadaException, FuncionSinRetornoException,
368     MalRetornoException {
369         returnFound = false;
370         for (NodoArbol actual : na.hijos) {
371             if (actual.REGLA_ID > 0) {
372                 switch (actual.REGLA_ID) {
373                     case ReglasProd.R_ASIGNACION:
374                         r_asig(ts, actual, tv, tf, ctx);
375                         break;
376                     case ReglasProd.R_CALL:
377                         r_llamada(ts, actual, tv, tf, ctx);
378                         break;
379                     case ReglasProd.R_IF:
380                         r_if(ts, actual, tv, tf, ctx);
381                         break;
382                     case ReglasProd.R_WHILE:
383                         r_while(ts, actual, tv, tf, ctx);
384                         break;
385                     case ReglasProd.R_FOR:
386                         r_for(ts, actual, tv, tf, ctx);
387                         break;
388                     case ReglasProd.R_LECTURA:
389                     case ReglasProd.R_IMPRESION:
390                         r_leer_imp(ts, actual, tv, tf, ctx);
391                 }
392             }
393         }
394     }

```

```

388                     break;
389
390         case ReglasProd.R_RETORNO:
391             if (ctx != null)
392                 r_retorno(ts, actual, tv, tf, ctx);
393             else
394                 throw new SentenciaInesperadaException(
395                         String.format("No se puede retornar un valor fuera de una
396                         función (línea %d).",
397                                     ts.getByID(actual.hijos.get(1).REF).LINE)
398                         );
399             returnFound = true;
400             break;
401         case ReglasProd.R_METODO:
402             throw new SentenciaInesperadaException("No se puede declarar una nueva
403             función dentro de un bloque.");
404         }
405     }
406     if (!returnFound && ctx != null) {
407         throw new FuncionSinRetornoException(
408             String.format("La función \"%s\" retorna un tipo %s, sin embargo no se encontró
409             la sentencia RETURN en su declaración.",
410             ctx.contexto, Tipos.nombres[tf.get(ctx.contexto).T_RETURN])
411         );
412     }
413
414 /**
415 * Método para verificar si una variable ya ha sido declarada
416 *
417 * @param nombre El nombre de la variable
418 * @param tv Tabla de nombres de variables
419 * @param ctx Función a la que pertenece dicha variable
420 * @return TRUE si la variable ha sido declarada
421 */

```

```
420     private static boolean variableDeclarada(String nombre, TablaVar tv, TablaVar ctx) {
421         return (ctx != null && ctx.contiene(nombre)) || tv.contiene(nombre);
422     }
423
424     /**
425      * Metodo con el cual obtendremos el tipo de Token a analizar
426      *
427      * @param ts Tabla de simbolos
428      * @param ref Referencia de la variable
429      * @return Retorna el tipo del token analizado
430      */
431     private static int getTipo(TablaSimbolos ts, String ref) {
432         RegistroTS reg = ts.getByID(ref);
433         int tipo = reg.TIPO;
434         if (tipo == Tipos.REF)
435             tipo = ts.getByID(reg.VAL).TIPO;
436         return tipo;
437     }
438
439     /**
440      * Metodo para identificar si el token es una palabra reservada
441      *
442      * @param tokenId La referencia en la tabla de simbolos
443      * @return regresara el Token ID si es una palabra reservada
444      */
445     private static boolean esReservadaTipo(int tokenId) {
446         return tokenId == Tokens.T_INT
447                 | tokenId == Tokens.T_REAL
448                 | tokenId == Tokens.T_BOOL
449                 | tokenId == Tokens.T_STRING;
450     }
451 }
```

```
1 package itc.automatas2.semantico;
2
3 import itc.automatas2.estructuras.*;
4 import itc.automatas2.semantico.excepciones.*;
5 import itc.automatas2.sintactico.ReglasProd;
6
7 import java.util.ArrayList;
8
9 /**
10 * Ejecuta el análisis semántico del programa haciendo uso del árbol sintáctico y la tabla de
11 * simbolos.
12 */
13 public class AnalizadorSemantico {
14     TablaVar vars;
15     TablaFunc funcs;
16
17     /**
18      * Hace el análisis semántico a partir de los árboles generados
19      * en el análisis sintáctico.
20      *
21      * @param ts      La tabla de simbolos,
22      * @param arboles Los árboles creados por el analizador sintáctico.
23      * @return <code>true</code> si el análisis no generó errores.
24
25     public boolean analizar(TablaSimbolos ts, ArrayList<ArbolSintactico> arboles) {
26         vars = new TablaVar();
27         funcs = new TablaFunc();
28         try {
29             for (ArbolSintactico a : arboles) {
30                 NodoArbol actual = a.raiz;
31                 if (actual.REGLA_ID > 0) {
32                     switch (actual.REGLA_ID) {
33                         case ReglasProd.R_METODO:
34                             ReglasSemanticas.r_func(ts, actual, vars, funcs,
35                                         new TablaVar(ts.getByID(actual.hijos.get(1).REF).NOMBRE)
```

```
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
```

```
        );
        break;
    case ReglasProd.R_ASIGNACION:
        ReglasSemanticas.r_asig(ts, actual, vars, funcs, null);
        break;
    case ReglasProd.R_CALL:
        ReglasSemanticas.r_llamada(ts, actual, vars, funcs, null);
        break;
    case ReglasProd.R_IF:
        ReglasSemanticas.r_if(ts, actual, vars, funcs, null);
        break;
    case ReglasProd.R_WHILE:
        ReglasSemanticas.r_while(ts, actual, vars, funcs, null);
        break;
    case ReglasProd.R_FOR:
        ReglasSemanticas.r_for(ts, actual, vars, funcs, null);
        break;
    case ReglasProd.R_LECTURA:
    case ReglasProd.R_IMPRESION:
        ReglasSemanticas.r_leer_imp(ts, actual, vars, funcs, null);
        break;
    }
}
}

} catch (VariableNoDeclaradaException e) {
    PilaErrores.meter(new RegistroErr(310, -1, e.getMessage()));
    return false;
} catch (TipoIncorrectoException e) {
    PilaErrores.meter(new RegistroErr(320, -1, e.getMessage()));
    return false;
} catch (ParametrosIncorrectosException e) {
    PilaErrores.meter(new RegistroErr(330, -1, e.getMessage()));
    return false;
} catch (IdExistenteException e) {
    PilaErrores.meter(new RegistroErr(340, -1, e.getMessage()));
}
```

```
70         return false;
71     } catch (MalRetornoException e) {
72         PilaErrores.meter(new RegistroErr(350, -1, e.getMessage()));
73         return false;
74     } catch (FuncionSinRetornoException e) {
75         PilaErrores.meter(new RegistroErr(360, -1, e.getMessage()));
76         return false;
77     } catch (SentenciaInesperadaException e) {
78         PilaErrores.meter(new RegistroErr(370, -1, e.getMessage()));
79         return false;
80     }
81     return true;
82 }
83 }
84 }
```

```
1 package itc.automatas2.semantico.excepciones;
2
3 public class MalRetornoException extends Exception {
4     public MalRetornoException() {
5         super();
6     }
7
8     public MalRetornoException(String message) {
9         super(message);
10    }
11 }
12
```

```
1 package itc.automatas2.semantico.excepciones;
2
3 public class IdExistenteException extends Exception {
4     public IdExistenteException() {
5         super();
6     }
7
8     public IdExistenteException(String message) {
9         super(message);
10    }
11 }
12
```

```
1 package itc.automatas2.semantico.excepciones;
2
3 public class TipoIncorrectoException extends Exception {
4     public TipoIncorrectoException() {
5         super();
6     }
7
8     public TipoIncorrectoException(String message) {
9         super(message);
10    }
11 }
12
```

```
1 package itc.automatas2.semantico.excepciones;
2
3 public class FuncionSinRetornoException extends Exception {
4     public FuncionSinRetornoException() {
5         super();
6     }
7
8     public FuncionSinRetornoException(String message) {
9         super(message);
10    }
11 }
12
```

```
1 package itc.automatas2.semantico.excepciones;
2
3 public class SentenciaInesperadaException extends Exception {
4     public SentenciaInesperadaException() {
5         super();
6     }
7
8     public SentenciaInesperadaException(String message) {
9         super(message);
10    }
11 }
12
```

```
1 package itc.automatas2.semantico.excepciones;
2
3 public class VariableNoDeclaradaException extends Exception {
4     public VariableNoDeclaradaException() {
5         super();
6     }
7
8     public VariableNoDeclaradaException(String message) {
9         super(message);
10    }
11 }
12
```

```
1 package itc.automatas2.semantico.excepciones;
2
3 public class ParametrosIncorrectosException extends Exception {
4     public ParametrosIncorrectosException() {
5         super();
6     }
7
8     public ParametrosIncorrectosException(String message) {
9         super(message);
10    }
11 }
12
```

Anexo

Código acumulado hasta esta fase

```
1 package itc.automatas2.gui;
2
3 import itc.automatas2.gui.controller.UIController;
4
5 import javax.swing.*;
6 import java.util.Locale;
7
8 public class Main {
9
10    public static void main(String[] args) {
11        try {
12            UIManager.setLookAndFeel(
13                UIManager.getSystemLookAndFeelClassName()
14            );
15            Locale.setDefault(new Locale("es", "MX"));
16            System.setProperty("awt.useSystemAAFontSettings", "on");
17            System.setProperty("swing.aatext", "true");
18        } catch (Exception e) {
19            e.printStackTrace();
20        }
21        SwingUtilities.invokeLater(() -> {
22            UIController controller = new UIController();
23        });
24    }
25 }
26 }
```

```
1 package itc.automatas2.gui.lib;
2
3 import javax.swing.*;
4 import javax.swing.table.TableCellRenderer;
5 import javax.swing.table.TableColumnModel;
6 import javax.xml.bind.DatatypeConverter;
7 import java.awt.*;
8 import java.security.MessageDigest;
9 import java.security.NoSuchAlgorithmException;
10
11 /**
12  * Clase de utilidades para el paquete.
13 */
14 public class Util {
15     /**
16      * Obtiene el hash de una cadena por medio del algoritmo MD5
17      *
18      * @param text la cadena a hashear
19      * @return el hash en formato hexadecimal
20      */
21     public static String md5(String text) {
22         if (text.isEmpty())
23             return "";
24         try {
25             MessageDigest md = MessageDigest.getInstance("MD5");
26             md.update(text.getBytes());
27             byte[] digest = md.digest();
28             return DatatypeConverter.printHexBinary(digest).toUpperCase();
29         } catch (NoSuchAlgorithmException e) {
30             e.printStackTrace();
31             return "";
32         }
33     }
34 }
35 /**

```

```
36     * Cambia el ancho de las columnas de una tabla de acuerdo a sus contenidos.
37     *
38     * @param table    una {@link JTable tabla}.
39     * @param minWidth el ancho mínimo que deben tener las columnas.
40     * @param padding  un espacio extra para dar al contenido más largo.
41     */
42    public static void autosizeColumns(JTable table, int minWidth, int padding) {
43        TableColumnModel model = table.getColumnModel();
44        for (int col = 0; col < table.getColumnCount(); col++) {
45            int width = minWidth;
46            for (int row = 0; row < table.getRowCount(); row++) {
47                TableCellRenderer renderer = table.getCellRenderer(row, col);
48                Component comp = table.prepareRenderer(renderer, row, col);
49                width = Math.max(comp.getPreferredSize().width + padding, width);
50            }
51            model.getColumn(col).setPreferredWidth(width);
52        }
53    }
54 }
```

A4 - MainFrame.java

```
1 package itc.automatas2.gui.view;
2
3 import org.fife.ui.rsyntaxtextarea.RSyntaxTextArea;
4 import org.fife.ui.rtextarea.RTextScrollPane;
5
6 import javax.swing.*;
7 import javax.swing.text.AttributeSet;
8 import javax.swing.text.BadLocationException;
9 import javax.swing.text.DefaultStyledDocument;
10
11 public class MainFrame extends javax.swing.JFrame {
12
13     public MainFrame() {
14         initComponents();
15     }
16
17     @SuppressWarnings("unchecked")
18     // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN:initComponents
19     private void initComponents() {
20
21         jToolBar1 = new javax.swing.JToolBar();
22         btnLex = new javax.swing.JButton();
23         btnSyn = new javax.swing.JButton();
24         btnSem = new javax.swing.JButton();
25         btnAll = new javax.swing.JButton();
26         filler1 = new javax.swing.Box.Filler(new java.awt.Dimension(0, 0), new java.awt.Dimension(0,
27             0), new java.awt.Dimension(32767, 0));
27         btnSTable = new javax.swing.JButton();
28         btnOutPane = new javax.swing.JButton();
29         jSplitPane = new javax.swing.JSplitPane();
30         rTextScrollPane = new org.fife.ui.rtextarea.RTextScrollPane();
31         rSyntaxTextAreal = new org.fife.ui.rsyntaxtextarea.RSyntaxTextArea();
32         jScrollPane2 = new javax.swing.JScrollPane();
33         txtOut = new javax.swing.JTextPane();
34         jMenuBar1 = new javax.swing.JMenuBar();
```

```
35      jMenu1 = new javax.swing.JMenu();
36      menuFileOpen = new javax.swing.JMenuItem();
37      menuFileSave = new javax.swing.JMenuItem();
38      menuFileSaveAs = new javax.swing.JMenuItem();
39      jSeparator1 = new javax.swing.JPopupMenu.Separator();
40      menuExit = new javax.swing.JMenuItem();
41      jMenu2 = new javax.swing.JMenu();
42      btnMenuAyudaLex = new javax.swing.JMenuItem();
43      btnMenuAyudaSyn = new javax.swing.JMenuItem();
44      btnMenuAyudaSem = new javax.swing.JMenuItem();
45
46      setDefaultCloseOperation(javax.swing.WindowConstants.DO NOTHING ON CLOSE);
47      setTitle("E1");
48      setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
49      setMinimumSize(new java.awt.Dimension(480, 360));
50      setPreferredSize(new java.awt.Dimension(800, 600));
51
52      jToolBar1.setFloatable(false);
53      jToolBar1.setRollover(true);
54      jToolBar1.setDoubleBuffered(true);
55
56      btnLex.setText("Léxico");
57      btnLex.setToolTipText("Análisis léxico (F5)");
58      btnLex.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.
createEtchedBorder(), null));
59      btnLex.setDoubleBuffered(true);
60      btnLex.setFocusable(false);
61      btnLex.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
62      btnLex.setMaximumSize(new java.awt.Dimension(80, 28));
63      btnLex.setMinimumSize(new java.awt.Dimension(80, 28));
64      btnLex.setPreferredSize(new java.awt.Dimension(80, 28));
65      btnLex.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
66      jToolBar1.add(btnLex);
67
68      btnSyn.setText("Sintáctico");
```

```
69         btnSyn.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.
 70             createEtchedBorder(), null));
 71         btnSyn.setDoubleBuffered(true);
 72         btnSyn.setFocusable(false);
 73         btnSyn.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
 74         btnSyn.setMaximumSize(new java.awt.Dimension(80, 28));
 75         btnSyn.setMinimumSize(new java.awt.Dimension(80, 28));
 76         btnSyn.setPreferredSize(new java.awt.Dimension(80, 28));
 77         btnSyn.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
 78         jToolBar1.add(btnSyn);
 79
 80         btnSem.setText("Semántico");
 81         btnSem.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.
 82             createEtchedBorder(), null));
 83         btnSem.setDoubleBuffered(true);
 84         btnSem.setFocusable(false);
 85         btnSem.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
 86         btnSem.setMaximumSize(new java.awt.Dimension(80, 28));
 87         btnSem.setMinimumSize(new java.awt.Dimension(80, 28));
 88         btnSem.setPreferredSize(new java.awt.Dimension(80, 28));
 89         btnSem.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
 90         jToolBar1.add(btnSem);
 91
 92         btnAll.setText("Todos");
 93         btnAll.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.
 94             createEtchedBorder(), null));
 95         btnAll.setDoubleBuffered(true);
 96         btnAll.setFocusable(false);
 97         btnAll.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
 98         btnAll.setMaximumSize(new java.awt.Dimension(80, 28));
 99         btnAll.setMinimumSize(new java.awt.Dimension(80, 28));
100         btnAll.setPreferredSize(new java.awt.Dimension(80, 28));
101         btnAll.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
102         jToolBar1.add(btnAll);
103         jToolBar1.add(filler1);
```

```
101      btnSTable.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.
102 BorderFactory.createEtchedBorder(), null));
103      btnSTable.setFocusable(false);
104      btnSTable.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
105      btnSTable.setLabel("Tabla de símbolos");
106      btnSTable.setMaximumSize(new java.awt.Dimension(120, 28));
107      btnSTable.setMinimumSize(new java.awt.Dimension(120, 28));
108      btnSTable.setPreferredSize(new java.awt.Dimension(120, 28));
109      btnSTable.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
110      jToolBar1.add(btnSTable);
111
112      btnOutPane.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.
113 BorderFactory.createEtchedBorder(), null));
114      btnOutPane.setFocusable(false);
115      btnOutPane.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
116      btnOutPane.setLabel("Salida");
117      btnOutPane.setMaximumSize(new java.awt.Dimension(60, 28));
118      btnOutPane.setMinimumSize(new java.awt.Dimension(60, 28));
119      btnOutPane.setPreferredSize(new java.awt.Dimension(48, 28));
120      btnOutPane.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
121      jToolBar1.add(btnOutPane);
122
123      getContentPane().add(jToolBar1, java.awt.BorderLayout.PAGE_START);
124
125      jSplitPane.setBorder(javax.swing.BorderFactory.createEmptyBorder(4, 4, 4, 4));
126      jSplitPane.setDividerLocation(350);
127      jSplitPane.setDividerSize(15);
128      jSplitPane.setOrientation(javax.swing.JSplitPane.VERTICAL_SPLIT);
129      jSplitPane.setResizeWeight(1.0);
130      jSplitPane.setAutoscrolls(true);
131      jSplitPane.setCursor(new java.awt.Cursor(java.awt.Cursor.N_RESIZE_CURSOR));
132      jSplitPane.setDoubleBuffered(true);
133      jSplitPane.setName(""); // NOI18N
```

```
134     rTextScrollPane1.setLineNumbersEnabled(true);  
135  
136     rSyntaxTextArea1.setColumns(20);  
137     rSyntaxTextArea1.setRows(5);  
138     rSyntaxTextArea1.setCodeFoldingEnabled(true);  
139     rSyntaxTextArea1.setFadeCurrentLineHighlight(true);  
140     rSyntaxTextArea1.setFont(new java.awt.Font("Consolas", 0, 12)); // NOI18N  
141     rSyntaxTextArea1.setMarginLineEnabled(true);  
142     rSyntaxTextArea1.setMarkOccurrences(true);  
143     rSyntaxTextArea1.setPaintMatchedBracketPair(true);  
144     rSyntaxTextArea1.setPaintTabLines(true);  
145     rSyntaxTextArea1.setTabsEmulated(true);  
146     rTextScrollPane1.setViewportView(rSyntaxTextArea1);  
147  
148     jSplitPane.setTopComponent(rTextScrollPane1);  
149  
150     jScrollPane2.setAutoscrolls(true);  
151     jScrollPane2.setDoubleBuffered(true);  
152     jScrollPane2.setMaximumSize(new java.awt.Dimension(32767, 400));  
153     jScrollPane2.setMinimumSize(new java.awt.Dimension(20, 0));  
154  
155     txtOut.setEditable(false);  
156     txtOut.setFont(new java.awt.Font("Monospaced", 0, 12)); // NOI18N  
157     txtOut.setToolTipText("");  
158     txtOut.setDoubleBuffered(true);  
159     txtOut.setMaximumSize(new java.awt.Dimension(2147483647, 400));  
160     jScrollPane2.setViewportView(txtOut);  
161  
162     jSplitPane.setRightComponent(jScrollPane2);  
163  
164     getContentPane().add(jSplitPane, java.awt.BorderLayout.CENTER);  
165  
166     jMenuBar1.setDoubleBuffered(true);  
167  
168     jMenuItem1.setText("Archivo");
```



```
200     btnMenuAyudaLex.setText("Análisis léxico");
201     jMenu2.add(btnMenuAyudaLex);
202
203     btnMenuAyudaSyn.setText("Análisis sintáctico");
204     jMenu2.add(btnMenuAyudaSyn);
205
206     btnMenuAyudaSem.setText("Análisis semántico");
207     jMenu2.add(btnMenuAyudaSem);
208
209     jMenuBar1.add(jMenu2);
210
211     setJMenuBar(jMenuBar1);
212
213     pack();
214 // </editor-fold> //GEN-END: initComponents
215
216     public JButton getBtnAll() {
217         return btnAll;
218     }
219
220     public JButton getBtnLex() {
221         return btnLex;
222     }
223
224     public JButton getBtnOutPane() {
225         return btnOutPane;
226     }
227
228     public JButton getBtnSTable() {
229         return btnSTable;
230     }
231
232     public JButton getBtnSem() {
233         return btnSem;
234     }
```

```
235
236     public JButton getBtnSyn() {
237         return btnSyn;
238     }
239
240     public JSplitPane getjSplitPane() {
241         return jSplitPane;
242     }
243
244     public JMenuItem getMenuExit() {
245         return menuExit;
246     }
247
248     public JMenuItem getMenuFileOpen() {
249         return menuFileOpen;
250     }
251
252     public JMenuItem getMenuFileSave() {
253         return menuFileSave;
254     }
255
256     public JMenuItem getMenuFileSaveAs() {
257         return menuFileSaveAs;
258     }
259
260     public JMenuItem getMenuAyudaLex() {
261         return btnMenuAyudaLex;
262     }
263
264     public JMenuItem getMenuAyudaSyn() {
265         return btnMenuAyudaSyn;
266     }
267
268     public JMenuItem getMenuAyudaSem() {
269         return btnMenuAyudaSem;
```

```
270     }
271
272     public JTextPane getTxtOut() {
273         return txtOut;
274     }
275
276     public RSyntaxTextArea getTxtCode() {
277         return rSyntaxTextArea1;
278     }
279
280     public RTextScrollPane getCodeScrollPane() {
281         return rTextScrollPane1;
282     }
283
284
285     // Variables declaration - do not modify//GEN-BEGIN:variables
286     private javax.swing.JButton btnAll;
287     private javax.swing.JButton btnLex;
288     private javax.swing.JMenuItem btnMenuAyudaLex;
289     private javax.swing.JMenuItem btnMenuAyudaSem;
290     private javax.swing.JMenuItem btnMenuAyudaSyn;
291     private javax.swing.JButton btnOutPane;
292     private javax.swing.JButton btnSTable;
293     private javax.swing.JButton btnSem;
294     private javax.swing.JButton btnSyn;
295     private javax.swing.Box.Filler filler1;
296     private javax.swing.JMenu jMenu1;
297     private javax.swing.JMenu jMenu2;
298     private javax.swing.JMenuBar jMenuBar1;
299     private javax.swing.JScrollPane jScrollPane2;
300     private javax.swing.JPopupMenu.Separator jSeparator1;
301     private javax.swing.JSplitPane jSplitPane;
302     private javax.swing.JToolBar jToolBar1;
303     private javax.swing.JMenuItem menuExit;
304     private javax.swing.JMenuItem menuFileOpen;
```

```
305     private javax.swing.JMenuItem menuFileSave;
306     private javax.swing.JMenuItem menuFileSaveAs;
307     private org.fife.ui.rsyntaxtextarea.RSyntaxTextArea rSyntaxTextArea1;
308     private org.fife.ui.rtextarea.RTextScrollPane rTextScrollPane1;
309     private javax.swing.JTextPane txtOut;
310     // End of variables declaration//GEN-END:variables
311
312     private static class TabDocument extends DefaultStyledDocument {
313
314         @Override
315         public void insertString(int offs, String str, AttributeSet a) throws BadLocationException
316         {
317             str = str.replaceAll("\t", "    ");
318             super.insertString(offs, str, a);
319         }
320     }
321 }
```

```
1 package itc.automatas2.gui.view;
2
3 import javax.swing.JTable;
4
5 public class TablaSimbolosDialog extends javax.swing.JFrame {
6
7     public TablaSimbolosDialog() {
8         initComponents();
9     }
10    @SuppressWarnings("unchecked")
11    // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN:initComponents
12    private void initComponents() {
13
14        jScrollPane2 = new javax.swing.JScrollPane();
15        symTable = new javax.swing.JTable();
16
17        setTitle("Tabla de símbolos");
18
19        jScrollPane2.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.
BorderFactory.createEtchedBorder(), javax.swing.BorderFactory.createEmptyBorder(4, 4, 4, 4)));
20        jScrollPane2.setPreferredSize(new java.awt.Dimension(480, 540));
21
22        symTable.setFont(new java.awt.Font("Consolas", 0, 12)); // NOI18N
23        symTable.setModel(new javax.swing.table.DefaultTableModel(
24            new Object [][] {
25
26                },
27                new String [] {
28                    "ID", "NOMBRE", "TOKEN_ID", "TIPO", "VALOR", "LINEA"
29                }
30            ) {
31                boolean[] canEdit = new boolean [] {
32                    false, false, false, false, false, false
33                };
34            
```

```
35         public boolean isCellEditable(int rowIndex, int columnIndex) {
36             return canEdit [columnIndex];
37         }
38     });
39     symTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
40     symTable.setFillsViewportHeight(true);
41     symTable.getTableHeader().setReorderingAllowed(false);
42     jScrollPane2.setViewportView(symTable);
43
44     getContentPane().add(jScrollPane2, java.awt.BorderLayout.CENTER);
45
46     pack();
47 } // </editor-fold> //GEN-END:initComponents
48
49 public JTable getSymTable() {
50     return symTable;
51 }
52
53 // Variables declaration - do not modify //GEN-BEGIN:variables
54 private javax.swing.JScrollPane jScrollPane2;
55 private javax.swing.JTable symTable;
56 // End of variables declaration //GEN-END:variables
57 }
58 }
```

```
1 package itc.automatas2.gui.model;
2
3 import java.io.*;
4
5 public class ArchivoModel {
6     private File file;
7     private String tmpPath;
8
9     public ArchivoModel(File file) {
10         this.file = file;
11         this.tmpPath = file.getPath() + ".tmp";
12         if (!file.exists()) {
13             try {
14                 file.createNewFile();
15             } catch (IOException e) {
16                 e.printStackTrace();
17             }
18         }
19     }
20
21     public File getFile() {
22         return file;
23     }
24
25     public String readAll() {
26         try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
27             StringBuilder sb = new StringBuilder();
28             reader.lines()
29                 .map(s -> s + "\n")
30                 .forEach(sb::append);
31             if (sb.length() > 1)
32                 sb.deleteCharAt(sb.length() - 1);
33             return sb.toString();
34         } catch (IOException e) {
35             e.printStackTrace();
36         }
37     }
38 }
```

```
36         return null;
37     }
38 }
39
40 public void write(String text) {
41     try {
42         File tmp = new File(tmpPath);
43         tmp.createNewFile();
44         BufferedWriter writer = new BufferedWriter(new PrintWriter(tmp));
45         writer.write(text);
46         writer.close();
47         file.delete();
48         tmp.renameTo(file);
49     } catch (IOException e) {
50         e.printStackTrace();
51     }
52 }
53 }
54 }
```

```
1 package itc.automatas2.gui.controller;
2
3 import itc.automatas2.estructuras.PilaErrores;
4 import itc.automatas2.gui.lib.Util;
5 import itc.automatas2.gui.model.ArchivoModel;
6 import itc.automatas2.gui.view.MainFrame;
7 import itc.automatas2.gui.view.TablaSimbolosDialog;
8 import itc.automatas2.lexico.AnalizadorLexico;
9 import itc.automatas2.lexico.Tipos;
10 import itc.automatas2.lexico.Tokens;
11 import itc.automatas2.semantico.AnalizadorSemantico;
12 import itc.automatas2.sintactico.AnalizadorSintactico;
13 import org.fife.ui.rsyntaxtextarea.AbstractTokenMakerFactory;
14 import org.fife.ui.rsyntaxtextarea.RSyntaxTextArea;
15 import org.fife.ui.rsyntaxtextarea.TokenMakerFactory;
16 import org.fife.ui.rsyntaxtextarea.folding.CurlyFoldParser;
17 import org.fife.ui.rsyntaxtextarea.folding.FoldParserManager;
18
19 import javax.swing.*;
20 import javax.swing.filechooser.FileNameExtensionFilter;
21 import javax.swing.table.DefaultTableModel;
22 import java.awt.*;
23 import java.awt.event.*;
24 import java.io.File;
25 import java.io.IOException;
26 import java.io.PrintStream;
27
28 /**
29  * Clase controladora de la ventana principal.
30 */
31 public class UIController {
32     private MainFrame frm;
33     private TablaSimbolosDialog dlgTable;
34     private JFileChooser fc;
35     private String digest;
```

```
36     private ArchivoModel handle;
37     private AnalizadorLexico al;
38     private AnalizadorSintactico aSi;
39     private AnalizadorSemantico aSe;
40
41
42     /**
43      * Constructor de la clase.
44      * Muestra la vista automáticamente.
45      */
46
47     public UIController() {
48         this.frm = new MainFrame();
49         this.C_DEFAULT = frm.getBtnLex().getBackground();
50         this.dlgTable = new TablaSimbolosDialog();
51         this.fc = new JFileChooser();
52         this.al = new AnalizadorLexico();
53         this.aSi = new AnalizadorSintactico();
54         this.aSe = new AnalizadorSemantico();
55         dlgTable.setLocationRelativeTo(frm);
56         initListeners();
57         configView();
58         setOutPaneVisible(false);
59         frm.setVisible(true);
60         digest = "";
61     }
62
63     /**
64      * Inicializa los listeners de la vista.
65      */
66
67     private void initListeners() {
68         // Frame
69         frm.addWindowListener(new WindowAdapter() {
70             @Override
71             public void windowClosing(WindowEvent e) {
72                 super.windowClosing(e);
73             }
74         });
75     }
76
77     /**
78      * Configura la vista.
79      */
80     private void configView() {
81         frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
82         frm.setTitle("Analizador Sintáctico");
83         frm.setExtendedState(JFrame.MAXIMIZED_BOTH);
84         frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
85         frm.setJMenuBar(menuBar);
86         frm.setJToolBar(toolbar);
87         frm.getContentPane().add(dlgTable, "Center");
88         frm.pack();
89     }
90
91     /**
92      * Muestra la vista automáticamente.
93      */
94     private void setOutPaneVisible(boolean visible) {
95         frm.setVisible(visible);
96     }
97
98     /**
99      * Devuelve el resultado de la ejecución.
100     */
101    public String getDigest() {
102        return digest;
103    }
104
105    /**
106     * Crea un nuevo analizador sintáctico.
107     */
108    public void crearAnalizadorSintactico() {
109        aSi = new AnalizadorSintactico();
110    }
111
112    /**
113     * Crea un nuevo analizador semántico.
114     */
115    public void crearAnalizadorSemantico() {
116        aSe = new AnalizadorSemantico();
117    }
118
119    /**
120     * Crea un nuevo analizador léxico.
121     */
122    public void crearAnalizadorLexico() {
123        al = new AnalizadorLexico();
124    }
125
126    /**
127     * Crea un nuevo archivo modelo.
128     */
129    public void crearArchivoModelo() {
130        handle = new ArchivoModel();
131    }
132
133    /**
134     * Crea un nuevo diálogo de tabla de simbólos.
135     */
136    public void crearTablaSimbolosDialog() {
137        dlgTable = new TablaSimbolosDialog();
138    }
139
140    /**
141     * Crea un nuevo explorador de archivos.
142     */
143    public void crearFileChooser() {
144        fc = new JFileChooser();
145    }
146
147    /**
148     * Crea una nueva barra de menú.
149     */
150    public void crearMenuBar() {
151        menuBar = new JMenuBar();
152    }
153
154    /**
155     * Crea una nueva barra de herramientas.
156     */
157    public void crearToolBar() {
158        toolbar = new JToolBar();
159    }
160
161    /**
162     * Crea una nueva ventana principal.
163     */
164    public void crearMainFrame() {
165        frm = new MainFrame();
166    }
167
168    /**
169     * Crea un nuevo adaptador de ventana.
170     */
171    public void crearWindowAdapter() {
172        frm.addWindowListener(new WindowAdapter() {
173            @Override
174            public void windowClosing(WindowEvent e) {
175                super.windowClosing(e);
176            }
177        });
178    }
179
180    /**
181     * Crea un nuevo analizador léxico.
182     */
183    public void crearAnalizadorLexico() {
184        al = new AnalizadorLexico();
185    }
186
187    /**
188     * Crea un nuevo analizador sintáctico.
189     */
190    public void crearAnalizadorSintactico() {
191        aSi = new AnalizadorSintactico();
192    }
193
194    /**
195     * Crea un nuevo analizador semántico.
196     */
197    public void crearAnalizadorSemantico() {
198        aSe = new AnalizadorSemantico();
199    }
200
201    /**
202     * Crea un nuevo diálogo de tabla de simbólos.
203     */
204    public void crearTablaSimbolosDialog() {
205        dlgTable = new TablaSimbolosDialog();
206    }
207
208    /**
209     * Crea un nuevo explorador de archivos.
210     */
211    public void crearFileChooser() {
212        fc = new JFileChooser();
213    }
214
215    /**
216     * Crea una nueva barra de menú.
217     */
218    public void crearMenuBar() {
219        menuBar = new JMenuBar();
220    }
221
222    /**
223     * Crea una nueva barra de herramientas.
224     */
225    public void crearToolBar() {
226        toolbar = new JToolBar();
227    }
228
229    /**
230     * Crea una nueva ventana principal.
231     */
232    public void crearMainFrame() {
233        frm = new MainFrame();
234    }
235
236    /**
237     * Crea un nuevo adaptador de ventana.
238     */
239    public void crearWindowAdapter() {
240        frm.addWindowListener(new WindowAdapter() {
241            @Override
242            public void windowClosing(WindowEvent e) {
243                super.windowClosing(e);
244            }
245        });
246    }
247
248    /**
249     * Crea un nuevo analizador léxico.
250     */
251    public void crearAnalizadorLexico() {
252        al = new AnalizadorLexico();
253    }
254
255    /**
256     * Crea un nuevo analizador sintáctico.
257     */
258    public void crearAnalizadorSintactico() {
259        aSi = new AnalizadorSintactico();
260    }
261
262    /**
263     * Crea un nuevo analizador semántico.
264     */
265    public void crearAnalizadorSemantico() {
266        aSe = new AnalizadorSemantico();
267    }
268
269    /**
270     * Crea un nuevo diálogo de tabla de simbólos.
271     */
272    public void crearTablaSimbolosDialog() {
273        dlgTable = new TablaSimbolosDialog();
274    }
275
276    /**
277     * Crea un nuevo explorador de archivos.
278     */
279    public void crearFileChooser() {
280        fc = new JFileChooser();
281    }
282
283    /**
284     * Crea una nueva barra de menú.
285     */
286    public void crearMenuBar() {
287        menuBar = new JMenuBar();
288    }
289
290    /**
291     * Crea una nueva barra de herramientas.
292     */
293    public void crearToolBar() {
294        toolbar = new JToolBar();
295    }
296
297    /**
298     * Crea una nueva ventana principal.
299     */
300    public void crearMainFrame() {
301        frm = new MainFrame();
302    }
303
304    /**
305     * Crea un nuevo adaptador de ventana.
306     */
307    public void crearWindowAdapter() {
308        frm.addWindowListener(new WindowAdapter() {
309            @Override
310            public void windowClosing(WindowEvent e) {
311                super.windowClosing(e);
312            }
313        });
314    }
315
316    /**
317     * Crea un nuevo analizador léxico.
318     */
319    public void crearAnalizadorLexico() {
320        al = new AnalizadorLexico();
321    }
322
323    /**
324     * Crea un nuevo analizador sintáctico.
325     */
326    public void crearAnalizadorSintactico() {
327        aSi = new AnalizadorSintactico();
328    }
329
330    /**
331     * Crea un nuevo analizador semántico.
332     */
333    public void crearAnalizadorSemantico() {
334        aSe = new AnalizadorSemantico();
335    }
336
337    /**
338     * Crea un nuevo diálogo de tabla de simbólos.
339     */
340    public void crearTablaSimbolosDialog() {
341        dlgTable = new TablaSimbolosDialog();
342    }
343
344    /**
345     * Crea un nuevo explorador de archivos.
346     */
347    public void crearFileChooser() {
348        fc = new JFileChooser();
349    }
350
351    /**
352     * Crea una nueva barra de menú.
353     */
354    public void crearMenuBar() {
355        menuBar = new JMenuBar();
356    }
357
358    /**
359     * Crea una nueva barra de herramientas.
360     */
361    public void crearToolBar() {
362        toolbar = new JToolBar();
363    }
364
365    /**
366     * Crea una nueva ventana principal.
367     */
368    public void crearMainFrame() {
369        frm = new MainFrame();
370    }
371
372    /**
373     * Crea un nuevo adaptador de ventana.
374     */
375    public void crearWindowAdapter() {
376        frm.addWindowListener(new WindowAdapter() {
377            @Override
378            public void windowClosing(WindowEvent e) {
379                super.windowClosing(e);
380            }
381        });
382    }
383
384    /**
385     * Crea un nuevo analizador léxico.
386     */
387    public void crearAnalizadorLexico() {
388        al = new AnalizadorLexico();
389    }
390
391    /**
392     * Crea un nuevo analizador sintáctico.
393     */
394    public void crearAnalizadorSintactico() {
395        aSi = new AnalizadorSintactico();
396    }
397
398    /**
399     * Crea un nuevo analizador semántico.
400     */
401    public void crearAnalizadorSemantico() {
402        aSe = new AnalizadorSemantico();
403    }
404
405    /**
406     * Crea un nuevo diálogo de tabla de simbólos.
407     */
408    public void crearTablaSimbolosDialog() {
409        dlgTable = new TablaSimbolosDialog();
410    }
411
412    /**
413     * Crea un nuevo explorador de archivos.
414     */
415    public void crearFileChooser() {
416        fc = new JFileChooser();
417    }
418
419    /**
420     * Crea una nueva barra de menú.
421     */
422    public void crearMenuBar() {
423        menuBar = new JMenuBar();
424    }
425
426    /**
427     * Crea una nueva barra de herramientas.
428     */
429    public void crearToolBar() {
430        toolbar = new JToolBar();
431    }
432
433    /**
434     * Crea una nueva ventana principal.
435     */
436    public void crearMainFrame() {
437        frm = new MainFrame();
438    }
439
440    /**
441     * Crea un nuevo adaptador de ventana.
442     */
443    public void crearWindowAdapter() {
444        frm.addWindowListener(new WindowAdapter() {
445            @Override
446            public void windowClosing(WindowEvent e) {
447                super.windowClosing(e);
448            }
449        });
450    }
451
452    /**
453     * Crea un nuevo analizador léxico.
454     */
455    public void crearAnalizadorLexico() {
456        al = new AnalizadorLexico();
457    }
458
459    /**
460     * Crea un nuevo analizador sintáctico.
461     */
462    public void crearAnalizadorSintactico() {
463        aSi = new AnalizadorSintactico();
464    }
465
466    /**
467     * Crea un nuevo analizador semántico.
468     */
469    public void crearAnalizadorSemantico() {
470        aSe = new AnalizadorSemantico();
471    }
472
473    /**
474     * Crea un nuevo diálogo de tabla de simbólos.
475     */
476    public void crearTablaSimbolosDialog() {
477        dlgTable = new TablaSimbolosDialog();
478    }
479
480    /**
481     * Crea un nuevo explorador de archivos.
482     */
483    public void crearFileChooser() {
484        fc = new JFileChooser();
485    }
486
487    /**
488     * Crea una nueva barra de menú.
489     */
490    public void crearMenuBar() {
491        menuBar = new JMenuBar();
492    }
493
494    /**
495     * Crea una nueva barra de herramientas.
496     */
497    public void crearToolBar() {
498        toolbar = new JToolBar();
499    }
500
501    /**
502     * Crea una nueva ventana principal.
503     */
504    public void crearMainFrame() {
505        frm = new MainFrame();
506    }
507
508    /**
509     * Crea un nuevo adaptador de ventana.
510     */
511    public void crearWindowAdapter() {
512        frm.addWindowListener(new WindowAdapter() {
513            @Override
514            public void windowClosing(WindowEvent e) {
515                super.windowClosing(e);
516            }
517        });
518    }
519
520    /**
521     * Crea un nuevo analizador léxico.
522     */
523    public void crearAnalizadorLexico() {
524        al = new AnalizadorLexico();
525    }
526
527    /**
528     * Crea un nuevo analizador sintáctico.
529     */
530    public void crearAnalizadorSintactico() {
531        aSi = new AnalizadorSintactico();
532    }
533
534    /**
535     * Crea un nuevo analizador semántico.
536     */
537    public void crearAnalizadorSemantico() {
538        aSe = new AnalizadorSemantico();
539    }
540
541    /**
542     * Crea un nuevo diálogo de tabla de simbólos.
543     */
544    public void crearTablaSimbolosDialog() {
545        dlgTable = new TablaSimbolosDialog();
546    }
547
548    /**
549     * Crea un nuevo explorador de archivos.
550     */
551    public void crearFileChooser() {
552        fc = new JFileChooser();
553    }
554
555    /**
556     * Crea una nueva barra de menú.
557     */
558    public void crearMenuBar() {
559        menuBar = new JMenuBar();
560    }
561
562    /**
563     * Crea una nueva barra de herramientas.
564     */
565    public void crearToolBar() {
566        toolbar = new JToolBar();
567    }
568
569    /**
570     * Crea una nueva ventana principal.
571     */
572    public void crearMainFrame() {
573        frm = new MainFrame();
574    }
575
576    /**
577     * Crea un nuevo adaptador de ventana.
578     */
579    public void crearWindowAdapter() {
580        frm.addWindowListener(new WindowAdapter() {
581            @Override
582            public void windowClosing(WindowEvent e) {
583                super.windowClosing(e);
584            }
585        });
586    }
587
588    /**
589     * Crea un nuevo analizador léxico.
590     */
591    public void crearAnalizadorLexico() {
592        al = new AnalizadorLexico();
593    }
594
595    /**
596     * Crea un nuevo analizador sintáctico.
597     */
598    public void crearAnalizadorSintactico() {
599        aSi = new AnalizadorSintactico();
600    }
601
602    /**
603     * Crea un nuevo analizador semántico.
604     */
605    public void crearAnalizadorSemantico() {
606        aSe = new AnalizadorSemantico();
607    }
608
609    /**
610     * Crea un nuevo diálogo de tabla de simbólos.
611     */
612    public void crearTablaSimbolosDialog() {
613        dlgTable = new TablaSimbolosDialog();
614    }
615
616    /**
617     * Crea un nuevo explorador de archivos.
618     */
619    public void crearFileChooser() {
620        fc = new JFileChooser();
621    }
622
623    /**
624     * Crea una nueva barra de menú.
625     */
626    public void crearMenuBar() {
627        menuBar = new JMenuBar();
628    }
629
630    /**
631     * Crea una nueva barra de herramientas.
632     */
633    public void crearToolBar() {
634        toolbar = new JToolBar();
635    }
636
637    /**
638     * Crea una nueva ventana principal.
639     */
640    public void crearMainFrame() {
641        frm = new MainFrame();
642    }
643
644    /**
645     * Crea un nuevo adaptador de ventana.
646     */
647    public void crearWindowAdapter() {
648        frm.addWindowListener(new WindowAdapter() {
649            @Override
650            public void windowClosing(WindowEvent e) {
651                super.windowClosing(e);
652            }
653        });
654    }
655
656    /**
657     * Crea un nuevo analizador léxico.
658     */
659    public void crearAnalizadorLexico() {
660        al = new AnalizadorLexico();
661    }
662
663    /**
664     * Crea un nuevo analizador sintáctico.
665     */
666    public void crearAnalizadorSintactico() {
667        aSi = new AnalizadorSintactico();
668    }
669
670    /**
671     * Crea un nuevo analizador semántico.
672     */
673    public void crearAnalizadorSemantico() {
674        aSe = new AnalizadorSemantico();
675    }
676
677    /**
678     * Crea un nuevo diálogo de tabla de simbólos.
679     */
680    public void crearTablaSimbolosDialog() {
681        dlgTable = new TablaSimbolosDialog();
682    }
683
684    /**
685     * Crea un nuevo explorador de archivos.
686     */
687    public void crearFileChooser() {
688        fc = new JFileChooser();
689    }
690
691    /**
692     * Crea una nueva barra de menú.
693     */
694    public void crearMenuBar() {
695        menuBar = new JMenuBar();
696    }
697
698    /**
699     * Crea una nueva barra de herramientas.
700     */
701    public void crearToolBar() {
702        toolbar = new JToolBar();
703    }
704
705    /**
706     * Crea una nueva ventana principal.
707     */
708    public void crearMainFrame() {
709        frm = new MainFrame();
710    }
711
712    /**
713     * Crea un nuevo adaptador de ventana.
714     */
715    public void crearWindowAdapter() {
716        frm.addWindowListener(new WindowAdapter() {
717            @Override
718            public void windowClosing(WindowEvent e) {
719                super.windowClosing(e);
720            }
721        });
722    }
723
724    /**
725     * Crea un nuevo analizador léxico.
726     */
727    public void crearAnalizadorLexico() {
728        al = new AnalizadorLexico();
729    }
730
731    /**
732     * Crea un nuevo analizador sintáctico.
733     */
734    public void crearAnalizadorSintactico() {
735        aSi = new AnalizadorSintactico();
736    }
737
738    /**
739     * Crea un nuevo analizador semántico.
740     */
741    public void crearAnalizadorSemantico() {
742        aSe = new AnalizadorSemantico();
743    }
744
745    /**
746     * Crea un nuevo diálogo de tabla de simbólos.
747     */
748    public void crearTablaSimbolosDialog() {
749        dlgTable = new TablaSimbolosDialog();
750    }
751
752    /**
753     * Crea un nuevo explorador de archivos.
754     */
755    public void crearFileChooser() {
756        fc = new JFileChooser();
757    }
758
759    /**
760     * Crea una nueva barra de menú.
761     */
762    public void crearMenuBar() {
763        menuBar = new JMenuBar();
764    }
765
766    /**
767     * Crea una nueva barra de herramientas.
768     */
769    public void crearToolBar() {
770        toolbar = new JToolBar();
771    }
772
773    /**
774     * Crea una nueva ventana principal.
775     */
776    public void crearMainFrame() {
777        frm = new MainFrame();
778    }
779
780    /**
781     * Crea un nuevo adaptador de ventana.
782     */
783    public void crearWindowAdapter() {
784        frm.addWindowListener(new WindowAdapter() {
785            @Override
786            public void windowClosing(WindowEvent e) {
787                super.windowClosing(e);
788            }
789        });
790    }
791
792    /**
793     * Crea un nuevo analizador léxico.
794     */
795    public void crearAnalizadorLexico() {
796        al = new AnalizadorLexico();
797    }
798
799    /**
800     * Crea un nuevo analizador sintáctico.
801     */
802    public void crearAnalizadorSintactico() {
803        aSi = new AnalizadorSintactico();
804    }
805
806    /**
807     * Crea un nuevo analizador semántico.
808     */
809    public void crearAnalizadorSemantico() {
810        aSe = new AnalizadorSemantico();
811    }
812
813    /**
814     * Crea un nuevo diálogo de tabla de simbólos.
815     */
816    public void crearTablaSimbolosDialog() {
817        dlgTable = new TablaSimbolosDialog();
818    }
819
820    /**
821     * Crea un nuevo explorador de archivos.
822     */
823    public void crearFileChooser() {
824        fc = new JFileChooser();
825    }
826
827    /**
828     * Crea una nueva barra de menú.
829     */
830    public void crearMenuBar() {
831        menuBar = new JMenuBar();
832    }
833
834    /**
835     * Crea una nueva barra de herramientas.
836     */
837    public void crearToolBar() {
838        toolbar = new JToolBar();
839    }
840
841    /**
842     * Crea una nueva ventana principal.
843     */
844    public void crearMainFrame() {
845        frm = new MainFrame();
846    }
847
848    /**
849     * Crea un nuevo adaptador de ventana.
850     */
851    public void crearWindowAdapter() {
852        frm.addWindowListener(new WindowAdapter() {
853            @Override
854            public void windowClosing(WindowEvent e) {
855                super.windowClosing(e);
856            }
857        });
858    }
859
860    /**
861     * Crea un nuevo analizador léxico.
862     */
863    public void crearAnalizadorLexico() {
864        al = new AnalizadorLexico();
865    }
866
867    /**
868     * Crea un nuevo analizador sintáctico.
869     */
870    public void crearAnalizadorSintactico() {
871        aSi = new AnalizadorSintactico();
872    }
873
874    /**
875     * Crea un nuevo analizador semántico.
876     */
877    public void crearAnalizadorSemantico() {
878        aSe = new AnalizadorSemantico();
879    }
880
881    /**
882     * Crea un nuevo diálogo de tabla de simbólos.
883     */
884    public void crearTablaSimbolosDialog() {
885        dlgTable = new TablaSimbolosDialog();
886    }
887
888    /**
889     * Crea un nuevo explorador de archivos.
890     */
891    public void crearFileChooser() {
892        fc = new JFileChooser();
893    }
894
895    /**
896     * Crea una nueva barra de menú.
897     */
898    public void crearMenuBar() {
899        menuBar = new JMenuBar();
900    }
901
902    /**
903     * Crea una nueva barra de herramientas.
904     */
905    public void crearToolBar() {
906        toolbar = new JToolBar();
907    }
908
909    /**
910     * Crea una nueva ventana principal.
911     */
912    public void crearMainFrame() {
913        frm = new MainFrame();
914    }
915
916    /**
917     * Crea un nuevo adaptador de ventana.
918     */
919    public void crearWindowAdapter() {
920        frm.addWindowListener(new WindowAdapter() {
921            @Override
922            public void windowClosing(WindowEvent e) {
923                super.windowClosing(e);
924            }
925        });
926    }
927
928    /**
929     * Crea un nuevo analizador léxico.
930     */
931    public void crearAnalizadorLexico() {
932        al = new AnalizadorLexico();
933    }
934
935    /**
936     * Crea un nuevo analizador sintáctico.
937     */
938    public void crearAnalizadorSintactico() {
939        aSi = new AnalizadorSintactico();
940    }
941
942    /**
943     * Crea un nuevo analizador semántico.
944     */
945    public void crearAnalizadorSemantico() {
946        aSe = new AnalizadorSemantico();
947    }
948
949    /**
950     * Crea un nuevo diálogo de tabla de simbólos.
951     */
952    public void crearTablaSimbolosDialog() {
953        dlgTable = new TablaSimbolosDialog();
954    }
955
956    /**
957     * Crea un nuevo explorador de archivos.
958     */
959    public void crearFileChooser() {
960        fc = new JFileChooser();
961    }
962
963    /**
964     * Crea una nueva barra de menú.
965     */
966    public void crearMenuBar() {
967        menuBar = new JMenuBar();
968    }
969
970    /**
971     * Crea una nueva barra de herramientas.
972     */
973    public void crearToolBar() {
974        toolbar = new JToolBar();
975    }
976
977    /**
978     * Crea una nueva ventana principal.
979     */
980    public void crearMainFrame() {
981        frm = new MainFrame();
982    }
983
984    /**
985     * Crea un nuevo adaptador de ventana.
986     */
987    public void crearWindowAdapter() {
988        frm.addWindowListener(new WindowAdapter() {
989            @Override
990            public void windowClosing(WindowEvent e) {
991                super.windowClosing(e);
992            }
993        });
994    }
995
996    /**
997     * Crea un nuevo analizador léxico.
998     */
999    public void crearAnalizadorLexico() {
1000        al = new AnalizadorLexico();
1001    }
1002
1003    /**
1004     * Crea un nuevo analizador sintáctico.
1005     */
1006    public void crearAnalizadorSintactico() {
1007        aSi = new AnalizadorSintactico();
1008    }
1009
1010    /**
1011     * Crea un nuevo analizador semántico.
1012     */
1013    public void crearAnalizadorSemantico() {
1014        aSe = new AnalizadorSemantico();
1015    }
1016
1017    /**
1018     * Crea un nuevo diálogo de tabla de simbólos.
1019     */
1020    public void crearTablaSimbolosDialog() {
1021        dlgTable = new TablaSimbolosDialog();
1022    }
1023
1024    /**
1025     * Crea un nuevo explorador de archivos.
1026     */
1027    public void crearFileChooser() {
1028        fc = new JFileChooser();
1029    }
1030
1031    /**
1032     * Crea una nueva barra de menú.
1033     */
1034    public void crearMenuBar() {
1035        menuBar = new JMenuBar();
1036    }
1037
1038    /**
1039     * Crea una nueva barra de herramientas.
1040     */
1041    public void crearToolBar() {
1042        toolbar = new JToolBar();
1043    }
1044
1045    /**
1046     * Crea una nueva ventana principal.
1047     */
1048    public void crearMainFrame() {
1049        frm = new MainFrame();
1050    }
1051
1052    /**
1053     * Crea un nuevo adaptador de ventana.
1054     */
1055    public void crearWindowAdapter() {
1056        frm.addWindowListener(new WindowAdapter() {
1057            @Override
1058            public void windowClosing(WindowEvent e) {
1059                super.windowClosing(e);
1060            }
1061        });
1062    }
1063
1064    /**
1065     * Crea un nuevo analizador léxico.
1066     */
1067    public void crearAnalizadorLexico() {
1068        al = new AnalizadorLexico();
1069    }
1070
1071    /**
1072     * Crea un nuevo analizador sintáctico.
1073     */
1074    public void crearAnalizadorSintactico() {
1075        aSi = new AnalizadorSintactico();
1076    }
1077
1078    /**
1079     * Crea un nuevo analizador semántico.
1080     */
1081    public void crearAnalizadorSemantico() {
1082        aSe = new AnalizadorSemantico();
1083    }
1084
1085    /**
1086     * Crea un nuevo diálogo de tabla de simbólos.
1087     */
1088    public void crearTablaSimbolosDialog() {
1089        dlgTable = new TablaSimbolosDialog();
1090    }
1091
1092    /**
1093     * Crea un nuevo explorador de archivos.
1094     */
1095    public void crearFileChooser() {
1096        fc = new JFileChooser();
1097    }
1098
1099    /**
1100     * Crea una nueva barra de menú.
1101     */
1102    public void crearMenuBar() {
1103        menuBar = new JMenuBar();
1104    }
1105
1106    /**
1107     * Crea una nueva barra de herramientas.
1108     */
1109    public void crearToolBar() {
1110        toolbar = new JToolBar();
1111    }
1112
1113    /**
1114     * Crea una nueva ventana principal.
1115     */
1116    public void crearMainFrame() {
1117        frm = new MainFrame();
1118    }
1119
1120    /**
1121     * Crea un nuevo adaptador de ventana.
1122     */
1123    public void crearWindowAdapter() {
1124        frm.addWindowListener(new WindowAdapter() {
1125            @Override
1126            public void windowClosing(WindowEvent e) {
1127                super.windowClosing(e);
1128            }
1129        });
1130    }
1131
1132    /**
1133     * Crea un nuevo analizador léxico.
1134     */
1135    public void crearAnalizadorLexico() {
1136        al = new AnalizadorLexico();
1137    }
1138
1139    /**
1140     * Crea un nuevo analizador sintáctico.
1141     */
1142    public void crearAnalizadorSintactico() {
1143        aSi = new AnalizadorSintactico();
1144    }
1145
1146    /**
1147     * Crea un nuevo analizador semántico.
1148     */
1149    public void crearAnalizadorSemantico() {
1150        aSe = new AnalizadorSemantico();
1151    }
1152
1153    /**
1154     * Crea un nuevo diálogo de tabla de simbólos.
1155     */
1156    public void crearTablaSimbolosDialog() {
1157        dlgTable = new TablaSimbolosDialog();
1158    }
1159
1160    /**
1161     * Crea un nuevo explorador de archivos.
1162     */
1163    public void crearFileChooser() {
1164        fc = new JFileChooser();
1165    }
1166
1167    /**
1168     * Crea una nueva barra de menú.
1169     */
1170    public void crearMenuBar() {
1171        menuBar = new JMenuBar();
1172    }
1173
1174    /**
1175     * Crea una nueva barra de herramientas.
1176     */
1177    public void crearToolBar() {
1178        toolbar = new JToolBar();
1179    }
1180
1181    /**
1182     * Crea una nueva ventana principal.
1183     */
1184    public void crearMainFrame() {
1185        frm = new MainFrame();
1186    }
1187
1188    /**
1189     * Crea un nuevo adaptador de ventana.
1190     */
1191    public void crearWindowAdapter() {
1192        frm.addWindowListener(new WindowAdapter() {
1193            @Override
1194            public void windowClosing(WindowEvent e) {
1195                super.windowClosing(e);
1196            }
1197        });
1198    }
1199
1200    /**
1201     * Crea un nuevo analizador léxico.
1202     */
1203    public void crearAnalizadorLexico() {
1204        al = new AnalizadorLexico();
1205    }
1206
1207    /**
1208     * Crea un nuevo analizador sintáctico.
1209     */
1210    public void crearAnalizadorSintactico() {
1211        aSi = new AnalizadorSintactico();
1212    }
1213
1214    /**
1215     * Crea un nuevo analizador semántico.
1216     */
1217    public void crearAnalizadorSemantico() {
1218        aSe = new AnalizadorSemantico();
1219    }
1220
1221    /**
1222     * Crea un nuevo diálogo de tabla de simbólos.
1223     */
1224    public void crearTablaSimbolosDialog() {
1225        dlgTable = new TablaSimbolosDialog();
1226    }
1227
1228    /**
1229     * Crea un nuevo explorador de archivos.
1230     */
1231    public void crearFileChooser() {
1232        fc = new JFileChooser();
1233    }
1234
1235    /**
1236     * Crea una nueva barra de menú.
1237     */
1238    public void crearMenuBar() {
1239        menuBar = new JMenuBar();
1240    }
1241
1242    /**
1243     * Crea una nueva barra de herramientas.
1244     */
1245    public void crearToolBar() {
1246        toolbar = new JToolBar();
1247    }
1248
1249    /**
1250     * Crea una nueva ventana principal.
1251     */
1252    public void crearMainFrame() {
1253        frm = new MainFrame();
1254    }
1255
1256    /**
1257     * Crea un nuevo adaptador de ventana.
1258     */
1259    public void crearWindowAdapter() {
1260        frm.addWindowListener(new WindowAdapter() {
1261            @Override
1262            public void windowClosing(WindowEvent e) {
1263                super.windowClosing(e);
1264            }
1265        });
1266    }
1267
1268    /**
1269     * Crea un nuevo analizador léxico.
1270     */
1271    public void crearAnalizadorLexico() {
1272        al = new AnalizadorLexico();
1273    }
1274
1275    /**
1276     * Crea un nuevo analizador sintáctico.
1277     */
1278    public void crearAnalizadorSintactico() {
1279        aSi = new AnalizadorSintactico();
1280    }
1281
1282    /**
1283     * Crea un nuevo analizador semántico.
1284     */
1285    public void crearAnalizadorSemantico() {
1286        aSe = new AnalizadorSemantico();
1287    }
1288
1289    /**
1290     * Crea un nuevo diálogo de tabla de simbólos.
1291     */
1292    public void crearTablaSimbolosDialog() {
1293        dlgTable = new TablaSimbolosDialog();
1294    }
1295
1296    /**
1297     * Crea un nuevo explorador de archivos.
1298     */
1299    public void crearFileChooser() {
1300        fc = new JFileChooser();
1301    }
1302
1303    /**
1304     * Crea una nueva barra de menú.
1305     */
1306    public void crearMenuBar() {
1307        menuBar = new JMenuBar();
1308    }
1309
1310    /**
1311     * Crea una nueva barra de herramientas.
1312     */
1313    public void crearToolBar() {
1314        toolbar = new JToolBar();
1315    }
1316
1317    /**
1318     * Crea una nueva ventana principal.
1319     */
1320    public void crearMainFrame() {
1321        frm = new MainFrame();
1322    }
1323
1324    /**
1325     * Crea un nuevo adaptador de ventana.
1326     */
1327    public void crearWindowAdapter() {
1328        frm.addWindowListener(new WindowAdapter() {
1329            @Override
1330            public void windowClosing(WindowEvent e) {
1331                super.windowClosing(e);
1332            }
1333        });
1334    }
1335
1336    /**
1337     * Crea un nuevo analizador léxico.
1338     */
1339    public void crearAnalizadorLexico() {
1340        al = new AnalizadorLexico();
1341    }
1342
1343    /**
1344     * Crea un nuevo analizador sintáctico.
1345     */
1346    public void crearAnalizadorSintactico() {
1347        aSi = new AnalizadorSintactico();
1348    }
1349
1350    /**
1351     * Crea un nuevo analizador semántico.
1352     */
1353    public void crearAnalizadorSemantico() {
1354        aSe = new AnalizadorSemantico();
1355    }
1356
1357    /**
1358     * Crea un nuevo diálogo de tabla de simbólos.
1359     */
1360    public void crearTablaSimbolosDialog() {
1361        dlgTable = new TablaSimbolosDialog();
1362    }
1363
1364    /**
1365     * Crea un nuevo explorador de archivos.
1366     */
1367    public void crearFileChooser() {
1368        fc =
```

```
71             salir();
72         }
73     } );
74
75     // Editor
76     frm.getTxtCode().addKeyListener(new KeyListener() {
77         @Override
78         public void keyTyped(KeyEvent e) {
79
80     }
81
82         @Override
83         public void keyPressed(KeyEvent e) {
84             switch (e.getKeyCode()) {
85                 case KeyEvent.VK_F5:
86                     if (frm.getBtnLex().isEnabled()) {
87                         if ((e.getModifiers() & ActionEvent.SHIFT_MASK) > 0)
88                             all();
89                         else
90                             lex(true);
91                     }
92                     break;
93                 case KeyEvent.VK_F6:
94                     if (frm.getBtnSyn().isEnabled())
95                         syn(true);
96                     }
97                     break;
98                 case KeyEvent.VK_F7:
99                     if (frm.getBtnSem().isEnabled())
100                         sem(true);
101                     }
102                     break;
103                 case KeyEvent.VK_T:
104                     if (frm.getBtnSTable().isEnabled())
105                         if ((e.getModifiers() & ActionEvent.CTRL_MASK) > 0)
```

```
106                     toggleTSDialog();
107                 }
108             }
109         }
110     }
111     @Override
112     public void keyReleased(KeyEvent e) {
113         }
114     }
115 } ;
116
117 // Menu archivo
118 frm.getMenuFileOpen().addActionListener(e -> abrirArchivo());
119 frm.getMenuFileSave().addActionListener(e -> guardarArchivo());
120 frm.getMenuFileSaveAs().addActionListener(e -> guardarComo());
121 frm.getMenuAyudaLex().addActionListener(e -> {
122     if (Desktop.isDesktopSupported()) {
123         try {
124             Desktop.getDesktop().open(new File("docs/lexico.pdf"));
125         } catch (IOException e1) {
126             e1.printStackTrace();
127         }
128     }
129 });
130 frm.getMenuAyudaSyn().addActionListener(e -> {
131     if (Desktop.isDesktopSupported()) {
132         try {
133             Desktop.getDesktop().open(new File("docs/sintactico.pdf"));
134         } catch (IOException e1) {
135             e1.printStackTrace();
136         }
137     }
138 });
139 frm.getMenuAyudaSem().addActionListener(e -> {
140     if (Desktop.isDesktopSupported()) {
```

```
141         try {
142             Desktop.getDesktop().open(new File("docs/semantico.pdf"));
143         } catch (IOException e1) {
144             e1.printStackTrace();
145         }
146     }
147 }
148 frm.getMenuExit().addActionListener(e -> salir());
149
150 // Toolbar
151 frm.getBtnLex().addActionListener(e -> lex(true));
152 frm.getBtnSyn().addActionListener(e -> syn(true));
153 frm.getBtnSem().addActionListener(e -> sem(true));
154 frm.getBtnAll().addActionListener(e -> all());
155 frm.getBtnSTable().addActionListener(e -> toggleTSDialog());
156 frm.getBtnOutPane().addActionListener(e -> toggleOutPane());
157 }
158
159 /**
160 * Configura los objetos de la vista.
161 */
162 private void configView() {
163     // Se redirigen las salidas al textpane
164     UIOutputStreamController out = new UIOutputStreamController(frm.getTxtOut());
165     UIErrorStreamController err = new UIErrorStreamController(frm.getTxtOut());
166     System.setOut(new PrintStream(out));
167     System.setErr(new PrintStream(err));
168
169     // Configuración del FileChooser
170     fc.setFileHidingEnabled(true);
171     fc.setAcceptAllFileFilterUsed(false);
172     fc.setMultiSelectionEnabled(false);
173     fc.setFileFilter(new FileNameExtensionFilter("Programa E1", "e1", "E1"));
174
175     // Botones
```

```
176         frm.getMenuFileSave().setEnabled(false);
177         btnState(S_INITIAL);
178         frm.getBtnSTable().setEnabled(false);
179
180         //Editor
181         AbstractTokenMakerFactory atmf = (AbstractTokenMakerFactory) TokenMakerFactory.
182             getDefaultInstance();
183         atmf.putMapping("text/E1", "itc.automatas2.gui.lib.E1TokenMaker");
184         FoldParserManager.get().addFoldParserMapping("text/E1", new CurlyFoldParser());
185         frm.getTxtCode().setSyntaxEditingStyle("text/E1");
186         frm.getTxtCodeScrollPane().setLineNumbersEnabled(true);
187         frm.getTxtCode().setCodeFoldingEnabled(true);
188     }
189
190     /**
191      * Abre un archivo utilizando un {@link JFileChooser FileChooser}.
192     */
193     private void abrirArchivo() {
194         fc.setDialogTitle("Abrir");
195         fc.setCurrentDirectory(new File("."));
196         if (fc.showOpenDialog(frm) == JFileChooser.APPROVE_OPTION) {
197             File file = fc.getSelectedFile();
198             handle = new ArchivoModel(file);
199             SwingUtilities.invokeLater(() -> {
200                 RSyntaxTextArea code = frm.getTxtCode();
201                 String content = handle.readAll();
202                 code.setText(content);
203                 digest = Util.md5(code.getText());
204                 code.discardAllEdits();
205                 frm.getMenuFileSave().setEnabled(true);
206                 updateTitle(handle.getFile());
207                 btnState(S_READY);
208                 frm.getBtnSTable().setEnabled(false);
209                 vaciarYOcultarTS();
210             });
211         }
212     }
```

```
210         }
211     }
212
213     /**
214      * Guarda el archivo actual.
215     */
216     private void guardarArchivo() {
217         SwingUtilities.invokeLater(() -> {
218             String content = frm.getTxtCode().getText();
219             handle.write(content);
220             digest = Util.md5(content);
221             btnState(S_READY);
222         });
223     }
224
225     /**
226      * Guarda un archivo utilizando un {@link JFileChooser FileChooser}.
227     */
228     private void guardarComo() {
229         fc.setDialogTitle("Guardar como");
230         if (fc.showSaveDialog(frm) == JFileChooser.APPROVE_OPTION) {
231             String path = fc.getSelectedFile().toString().endsWith(".el")
232                 ? fc.getSelectedFile().toString()
233                 : fc.getSelectedFile().toString() + ".el";
234             File file = new File(path);
235             handle = new ArchivoModel(file);
236             SwingUtilities.invokeLater(() -> {
237                 if (codeChanged()) {
238                     String content = frm.getTxtCode().getText();
239                     handle.write(content);
240                     digest = Util.md5(content);
241                 }
242                 frm.getMenuFileSave().setEnabled(true);
243                 updateTitle(handle.getFile());
244                 btnState(S_READY);
245             });
246         }
247     }
248 }
```

```
245             frm.getBtnSTable().setEnabled(false);
246         });
247     }
248
249 }
250
251 /**
252 * Maneja el cierre del programa y los cambios al código.
253 */
254 private void salir() {
255     if (codeChanged()) {
256         int option = JOptionPane.showConfirmDialog(frm,
257             "¿Desea guardar los cambios?", "Salir",
258             JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE
259         );
260         switch (option) {
261             case JOptionPane.YES_OPTION:
262                 guardarArchivo();
263             case JOptionPane.NO_OPTION:
264                 System.exit(0);
265                 break;
266             case JOptionPane.CANCEL_OPTION:
267             case JOptionPane.CLOSED_OPTION:
268                 System.out.println("Cancelled");
269                 break;
270         }
271     } else System.exit(0);
272 }
273
274 /**
275 * Muestra u oculta el panel de la salida del programa.
276 *
277 * @param visible la visibilidad del panel.
278 */
279 private void setOutPaneVisible(boolean visible) {
```

```
280         SwingUtilities.invokeLater(() -> {
281             JSplitPane pane = frm.getjSplitPane();
282             if (!visible) {
283                 pane.setDividerLocation(1.0d);
284                 frm.getTxtOut().setText("");
285             } else {
286                 pane.setDividerLocation(pane.getHeight() - 200);
287             }
288             pane.setEnabled(visible);
289         });
290     }
291
292     /**
293      * Alterna la visibilidad del panel de salida.
294      */
295     private void toggleOutPane() {
296         setOutPaneVisible(!frm.getjSplitPane().isEnabled());
297     }
298
299     /**
300      * Alerta al usuario que hubo cambios al código.
301      */
302     private void promptChanged() {
303         JOptionPane.showMessageDialog(frm,
304             "Los cambios deben ser guardados para proceder con el análisis.\n" +
305             "Por favor, guarde y vuelva a intentar.", "Advertencia",
306             JOptionPane.WARNING_MESSAGE);
307     }
308
309     /**
310      * Ejecuta el análisis léxico del programa actual.
311      */
312     private void lex(boolean imprimirErrores) {
313         if (codeChanged()) {
314             promptChanged();
```

```
315         } else if (handle != null) {
316             if (!frm.getjSplitPane().isEnabled())
317                 setOutPaneVisible(true);
318             System.out.printf("ANÁLISIS LÉXICO DEL PROGRAMA \"%s\"\n", handle.getFile().toString());
319             if (al.analizar(handle.getFile().toString())) {
320                 System.out.println("El analizador léxico declaró el código como válido");
321                 btnState(S_LEX_GOOD);
322             } else {
323                 System.err.println("El analizador léxico declaró el código como inválido");
324                 btnState(S_LEX_ERR);
325             }
326             poblarTS();
327             if (imprimirErrores)
328                 PilaErrores.imprimirErrores();
329             frm.getBtnSTable().setEnabled(true);
330             System.out.println();
331         }
332     }
333
334 /**
335 * Ejecuta el análisis sintáctico del programa actual.
336 */
337 private void syn(boolean imprimirErrores) {
338     if (codeChanged())
339         promptChanged();
340     else if (al.tS.size() > 0) {
341         if (!frm.getjSplitPane().isEnabled())
342             setOutPaneVisible(true);
343         System.out.printf("ANÁLISIS SINTÁCTICO DEL PROGRAMA \"%s\"\n", handle.getFile().toString
());
344         if (aSi.analizar(al.tS)) {
345             System.out.println("El analizador sintáctico declaró el código como válido");
346             btnState(S_SYN_GOOD);
347         } else {
348             System.err.println("El analizador sintáctico declaró el código como inválido");
```

```
349             btnState(S_SYN_ERR);
350         }
351         if (aSi.tieneArboles()) {
352             System.out.println("Árboles construidos:");
353             aSi.imprimirArboles();
354         }
355         poblarTS();
356         if (imprimirErrores)
357             PilaErrores.imprimirErrores();
358         System.out.println();
359     }
360 }
361 /**
362 * Ejecuta el análisis semántico del programa actual.
363 */
364 private void sem(boolean imprimirErrores) {
365     if (codeChanged())
366         promptChanged();
367     } else if (aSi.tieneArboles())
368         if (!frm.getjSplitPane().isEnabled())
369             setOutPaneVisible(true);
370         System.out.printf("ANÁLISIS SINTÁCTICO DEL PROGRAMA \"%s\"\n", handle.getFile().toString
());
371
372         if (aSe.analizar(al.ts, aSi.arboles)) {
373             System.out.println("El analizador semántico declaró el código como válido");
374             btnState(S_SEM_GOOD);
375         } else {
376             System.err.println("El analizador semántico declaró el código como inválido");
377             btnState(S_SEM_ERR);
378         }
379         if (imprimirErrores)
380             PilaErrores.imprimirErrores();
381         System.out.println();
382 }
```

```
383     }
384
385     /**
386      * Ejecuta todos los análisis en cadena.
387      */
388     private void all() {
389         if (codeChanged()) {
390             promptChanged();
391         } else {
392             lex(false);
393             syn(false);
394             sem(false);
395             PilaErrores.imprimirErrores();
396         }
397     }
398
399
400     /**
401      * Limpia la tabla de símbolos y oculta su ventana.
402      */
403     private void vaciarYOcultarTS() {
404         SwingUtilities.invokeLater(() -> {
405             DefaultTableModel dtm = (DefaultTableModel) dlgTable.getSymTable().getModel();
406             dtm.setRowCount(0);
407             dlgTable.setVisible(false);
408         });
409     }
410
411     /**
412      * Llena la tabla de símbolos.
413      */
414     private void poblarTS() {
415         SwingUtilities.invokeLater(() -> {
416             DefaultTableModel dtm = (DefaultTableModel) dlgTable.getSymTable().getModel();
417             dtm.setRowCount(0);
```

```
418         al.tS.registros().forEach(reg -> dtm.addRow(new Object[] {
419             reg.ID,
420             reg.NOMBRE,
421             Tokens.nombres[reg.TOKEN_ID],
422             Tipos.nombres[reg.TIPO],
423             reg.VAL,
424             reg.LINE
425         }));
426         Util.autosizeColumns(dlgTable.getSymTable(), 72, 16);
427     });
428 }
429
430 /**
431 * Alterna la visibilidad de la tabla de símbolos.
432 */
433 private void toggleTSDialog() {
434     dlgTable.setVisible(!dlgTable.isVisible());
435 }
436
437 /**
438 * Actualiza el título de la ventana a partir del nombre del archivo activo.
439 *
440 * @param activeFile el archivo activo.
441 */
442 private void updateTitle(File activeFile) {
443     frm.setTitle(String.format("%s [%s]", "E1", activeFile));
444 }
445
446
447 /**
448 * Verifica si hubo cambios en el código.
449 *
450 * @return <code>true</code> si se hicieron cambios.
451 */
452 private boolean codeChanged() {
```

```
453     String newDigest = Util.md5(frm.getTxtCode().getText());
454     return !digest.equals(newDigest);
455 }
456
457 private void btnState(int state) {
458     SwingUtilities.invokeLater(() -> {
459         frm.getBtnLex().setEnabled(state >= S_READY);
460         frm.getBtnSyn().setEnabled(state >= S_LEX_GOOD);
461         frm.getBtnSem().setEnabled(state >= S_SYN_GOOD);
462         frm.getBtnAll().setEnabled(state >= S_READY);
463         switch (state) {
464             case S_INITIAL:
465                 frm.getBtnLex().setBackground(C_DEFAULT);
466                 frm.getBtnSyn().setBackground(C_DEFAULT);
467                 frm.getBtnSem().setBackground(C_DEFAULT);
468                 break;
469             case S_READY:
470                 frm.getBtnLex().setBackground(C_YELLOW);
471                 frm.getBtnSyn().setBackground(C_DEFAULT);
472                 frm.getBtnSem().setBackground(C_DEFAULT);
473                 break;
474             case S_LEX_ERR:
475                 frm.getBtnLex().setBackground(C_RED);
476                 break;
477             case S_LEX_GOOD:
478                 frm.getBtnLex().setBackground(C_GREEN);
479                 frm.getBtnSyn().setBackground(C_YELLOW);
480                 break;
481             case S_SYN_ERR:
482                 frm.getBtnSyn().setBackground(C_RED);
483                 break;
484             case S_SYN_GOOD:
485                 frm.getBtnSyn().setBackground(C_GREEN);
486                 frm.getBtnSem().setBackground(C_YELLOW);
487                 break;
488         }
489     });
490 }
```

```
488         case S_SEM_ERR:  
489             frm.getBtnLex().setBackground(C_GREEN);  
490             frm.getBtnSyn().setBackground(C_GREEN);  
491             frm.getBtnSem().setBackground(C_RED);  
492             break;  
493         case S_SEM_GOOD:  
494             frm.getBtnLex().setBackground(C_GREEN);  
495             frm.getBtnSyn().setBackground(C_GREEN);  
496             frm.getBtnSem().setBackground(C_GREEN);  
497             break;  
498     }  
499 } ;  
500 }  
501  
502  
503     private static final int S_INITIAL = 0;  
504     private static final int S_READY = 10;  
505     private static final int S_LEX_ERR = 11;  
506     private static final int S_LEX_GOOD = 12;  
507     private static final int S_SYN_ERR = 21;  
508     private static final int S_SYN_GOOD = 22;  
509     private static final int S_SEM_ERR = 31;  
510     private static final int S_SEM_GOOD = 32;  
511  
512     private final Color C_RED = new Color(255, 105, 97);  
513     private final Color C_YELLOW = new Color(253, 253, 150);  
514     private final Color C_GREEN = new Color(119, 221, 119);  
515     private final Color C_DEFAULT;  
516 }  
517
```

```
1 package itc.automatas2.gui.controller;
2
3 import javax.swing.*;
4 import javax.swing.text.BadLocationException;
5 import javax.swing.text.Document;
6 import javax.swing.text.SimpleAttributeSet;
7 import javax.swing.text.StyleConstants;
8 import java.awt.*;
9 import java.io.OutputStream;
10 import java.io.PrintStream;
11
12 public class UIErrorStreamController extends OutputStream {
13     private final PrintStream STDERR = System.err;
14     private Document doc;
15     private SimpleAttributeSet as;
16     byte last;
17
18     public UIErrorStreamController(JTextPane textPane) {
19         this.doc = textPane.getDocument();
20         as = new SimpleAttributeSet();
21         StyleConstants.setForeground(as, Color.RED);
22     }
23
24     @Override
25     public void write(int b) {
26         //STDERR.write(b);
27         if (b < 0 && last == 0) {
28             last = (byte) b;
29         } else {
30             byte chr[];
31             if (last != 0) {
32                 chr = new byte[] {
33                     last, (byte) b
34                 };
35             }
36             last = 0;
```

```
36     } else {
37         chr = new byte[] {
38             (byte) b
39         };
40     }
41     String c = new String(chr);
42     SwingUtilities.invokeLater(() -> {
43         try {
44             doc.insertString(doc.getLength(), c, as);
45         } catch (BadLocationException e) {
46             STDERR.println(e);
47         }
48     });
49 }
50 }
51 }
52 }
```

```
1 package itc.automatas2.gui.controller;
2
3 import javax.swing.*;
4 import javax.swing.text.BadLocationException;
5 import javax.swing.text.Document;
6 import javax.swing.text.SimpleAttributeSet;
7 import javax.swing.text.StyleConstants;
8 import java.awt.*;
9 import java.io.ByteArrayOutputStream;
10 import java.io.OutputStream;
11 import java.io.PrintStream;
12 import java.util.ArrayList;
13
14 public class UIOutputStreamController extends OutputStream {
15     private final PrintStream STDOUT = System.out;
16     private final PrintStream STDERR = System.err;
17     private Document doc;
18     private SimpleAttributeSet as;
19     private byte last;
20
21     public UIOutputStreamController(JTextPane textPane) {
22         this.doc = textPane.getDocument();
23         as = new SimpleAttributeSet();
24         StyleConstants.setForeground(as, Color.BLACK);
25     }
26
27     @Override
28     public void write(int b) {
29         // STDOUT.write(b);
30         if (b < 0 && last == 0) {
31             last = (byte) b;
32         } else {
33             byte chr[];
34             if (last != 0) {
35                 chr = new byte[] {
```

```
36                     last, (byte) b
37                 } ;
38                 last = 0;
39             }
40         else {
41             chr = new byte[] {
42                 (byte) b
43             } ;
44         }
45         String c = new String(chr);
46         SwingUtilities.invokeLater(() -> {
47             try {
48                 doc.insertString(doc.getLength(), c, as);
49             } catch (BadLocationException e) {
50                 STDERR.println(e);
51             }
52         } );
53     }
54 }
55 }
56 }
```

```
1 package itc.automatas2.misc;
2
3
4 /**
5  * TDA para contener la descripción de un error.
6 */
7 public class Error {
8     public final int ID;
9     public final String DESC;
10    public String REASON;
11
12
13    Error(int ID, String DESC) {
14        this.ID = ID;
15        this.DESC = DESC;
16        this.REASON = "";
17    }
18
19    /**
20     * Establece la razón del error. Útil para imprimirlo en pantalla.
21     *
22     * @param reason La razón del error.
23     * @return El mismo error, por conveniencia.
24     */
25    public Error setReason(String reason) {
26        this.REASON = reason;
27        return this;
28    }
29
30    /**
31     * @return Una representación del error de la forma "ID: DESC REASON"
32     */
33    @Override
34    public String toString() {
35        return String.format("ERROR %d: %s %s", ID, DESC, REASON);
```

```
36      }
37  }
38
```

```
1 package itc.automatas2.misc;
2
3 import java.util.Hashtable;
4
5 /**
6  * Catálogo de errores internos. Esta clase contiene la definición de cada error posible.
7 */
8 public class BaseErrores {
9     private static Hashtable<Integer, Error> tablaErrores = new Hashtable<>();
10
11     static {
12         //Errores reservados para el sistema
13         tablaErrores.put(10, new Error(10, "El archivo al que se hace referencia no existe."));
14         tablaErrores.put(11, new Error(11, "Ocurrió un error de E/S al leer el código fuente."));
15         tablaErrores.put(12, new Error(12, "Programa incompleto."));
16
17         //Errores lexicos
18         tablaErrores.put(110, new Error(110, "Token inválido."));
19         tablaErrores.put(120, new Error(120, "Uso de símbolos no definidos en el alfabeto."));
20
21         //Errores sintacticos
22         tablaErrores.put(210, new Error(210, "Falta el delimitador ';' al final de la sentencia."));
23         tablaErrores.put(220, new Error(220, "Return sólo acepta un parámetro seguido de ';'."));
24         tablaErrores.put(230, new Error(230, "Apertura o cierre de llaves o corchetes erróneas."));
25         tablaErrores.put(240, new Error(240, "Mala estructuración de sentencias según el lenguaje."));
26     };
27
28         tablaErrores.put(250, new Error(250, "Estructura no reconocida por el lenguaje."));
29
30         //Errores semánticos
31         tablaErrores.put(310, new Error(310, "Identificador no declarado."));
32         tablaErrores.put(320, new Error(320, "Error de tipos de dato."));
33         tablaErrores.put(330, new Error(330, "Pase de parámetros incorrecto."));
34         tablaErrores.put(340, new Error(340, "Identificador ya declarado."));
35         tablaErrores.put(350, new Error(350, "Retorno incorrecto."));
36         tablaErrores.put(360, new Error(360, "Función sin retorno."));
```

```
35         tablaErrores.put(370, new Error(370, "Sentencia inesperada."));  
36     }  
37  
38     /**  
39      * Obtiene un objeto {@link Error Error} de acuerdo al ID proporcionado.  
40      *  
41      * @param ID El identificador del error.  
42      * @return Un objeto {@link Error Error} si existe, <code>null</code> si no.  
43      */  
44     public static Error getError(int ID) {  
45         return tablaErrores.get(ID);  
46     }  
47 }
```

```
1 package itc.automatas2.lexico;
2
3 /**
4  * Catálogo de tipos de datos
5 */
6 public class Tipos {
7     public static final int VOID = 0;
8     public static final int INT = 1;
9     public static final int REAL = 2;
10    public static final int BOOL = 3;
11    public static final int STR = 4;
12    public static final int REF = 5;
13
14    public static final String[] nombres = {
15        "VOID",
16        "INT",
17        "REAL",
18        "BOOL",
19        "STR",
20        "REF"
21    };
22 }
23
```

```
1 package itc.automatas2.lexico;
2
3 /**
4  * Catálogo de tokens e IDs.
5 */
6 public class Tokens {
7
8     //Palabras reservadas
9     public static final int T_IF = 1;
10    public static final int T_ELSE = 2;
11    public static final int T_WHILE = 3;
12    public static final int T_FOR = 4;
13    public static final int T_IN = 5;
14    public static final int T_FUNC = 6;
15    public static final int T_RETURN = 7;
16    public static final int T_TRUE = 8;
17    public static final int T_FALSE = 9;
18    public static final int T_NULL = 10;
19    public static final int T_INT = 11;
20    public static final int T_REAL = 12;
21    public static final int T_BOOL = 13;
22    public static final int T_STRING = 14;
23    public static final int T_PRINT = 15;
24    public static final int T_PRINTLN = 16;
25    public static final int T_READ = 17;
26
27     //Identificadores y constantes
28    public static final int T_VAR = 18;
29    public static final int T_FUN = 19;
30    public static final int T_INT_CONST = 20;
31    public static final int T_REAL_CONST = 21;
32    public static final int T_BOOL_CONST = 22;
33    public static final int T_STR_CONST = 23;
34
35     //Símbolos y operadores
```

```
36     public static final int T_COLON = 24;
37     public static final int T_SEMICOLON = 25;
38     public static final int T_LPAREN = 26;
39     public static final int T_RPAREN = 27;
40     public static final int T_LBRACE = 28;
41     public static final int T_RBRACE = 29;
42     public static final int T_LBRACKET = 30;
43     public static final int T_RBRACKET = 31;
44     public static final int T_LTE = 32;
45     public static final int T_GTE = 33;
46     public static final int T_NE = 34;
47     public static final int T_LT = 35;
48     public static final int T_GT = 36;
49     public static final int T_EQ = 37;
50     public static final int T_ASSIGN = 38;
51     public static final int T_DEC = 39;
52     public static final int T_INC = 40;
53     public static final int T_NOT = 41;
54     public static final int T_PLUS = 42;
55     public static final int T_MINUS = 43;
56     public static final int T_STAR = 44;
57     public static final int T_DIV = 45;
58     public static final int T_RANGE = 46;
59     public static final int T_COMMA = 47;
60
61     public static final String[] nombres = {
62         "UNDEFINED",
63         "T_IF",
64         "T_ELSE",
65         "T_WHILE",
66         "T_FOR",
67         "T_IN",
68         "T_FUNC",
69         "T_RETURN",
70         "T_TRUE",
```

```
71     "T_FALSE",
72     "T_NULL",
73     "T_INT",
74     "T_REAL",
75     "T_BOOL",
76     "T_STRING",
77     "T_PRINT",
78     "T_PRINTLN",
79     "T_READ",
80     "T_VAR",
81     "T_FUN",
82     "T_INT_CONST",
83     "T_REAL_CONST",
84     "T_BOOL_CONST",
85     "T_STR_CONST",
86     "T_COLON",
87     "T_SEMICOLON",
88     "T_LPAREN",
89     "T_RPAREN",
90     "T_LBRACE",
91     "T_RBRACE",
92     "T_LBRACKET",
93     "T_RBRACKET",
94     "T_LTE",
95     "T_GTE",
96     "T_NE",
97     "T_LT",
98     "T_GT",
99     "T_EQ",
100    "T_ASSIGN",
101    "T_DEC",
102    "T_INC",
103    "T_NOT",
104    "T_PLUS",
105    "T_MINUS",
```

```
106     "T_STAR",
107     "T_DIV",
108     "T_RANGE",
109     "T_COMMA"
110   } ;
111 }
```

```
1 package itc.automatas2.lexico;
2
3 import itc.automatas2.estructuras.Archivo;
4
5 import java.io.FileNotFoundException;
6 import java.io.IOException;
7 import java.util.ArrayDeque;
8
9 /**
10  * Clase que genera los tokens que seran utilizados por el analizador.
11 */
12 public class Tokenizador {
13     public Archivo archivo;
14     private ArrayDeque<String> tokens;
15
16     /**
17      * Constructor de la clase
18      *
19      * @param ruta
20      * @throws FileNotFoundException
21      */
22     public Tokenizador(String ruta) throws FileNotFoundException {
23         archivo = new Archivo(ruta);
24         tokens = new ArrayDeque<>();
25     }
26
27     /**
28      * Obtiene un token de la cola que la misma clase mantiene.
29      *
30      * @return Un token de la cola, <code>null</code> si se alcanzó el final del archivo.
31      * @throws IOException Si ocurre un error de lectura del archivo.
32      */
33     public String siguienteToken() throws IOException {
34         if (tieneTokens()) {
35             return tokens.remove();
```

```
36         }
37
38         return null;
39     }
40
41     /**
42      * Se salta a la siguiente linea, ignorando los tokens que existan en la cola.
43      */
44     public void siguienteLinea() {
45         tokens.clear();
46     }
47
48     /**
49      * El metodo utiliza la linea que la clase Archivo le envia, separa la linea en tokens usando el
50      * espacio como delimitador.
51      *
52      * @return Regresa VERDADERO si aun hay tokens y regresa FALSO si ya no hay tokens con los que
53      * trabajar.
54      * @throws IOException Si ocurre un error en la lectura o el archivo ya se cerró.
55      */
56     private boolean tieneTokens() throws IOException {
57         if (tokens.isEmpty()) {//Si tokens esta vacia se llenara con tokens
58             String linea = archivo.leerLinea();
59             while (linea != null && linea.trim().length() == 0) {
60                 linea = archivo.leerLinea();
61             }
62
63             if (linea == null)
64                 return false;
65
66             //Simbolo especial
67             boolean special = false;
68             char lastSpecial = 0;
69             //Bandera para saber si se está leyendo una cadena
70             boolean string = false;
```

```
69
70     StringBuilder sb = new StringBuilder();
71     char[] str = linea.toCharArray();
72     for (int i = 0; i < str.length; i++) {
73         if (!string) { //No se está leyendo un string
74             switch (str[i]) {
75                 //Delimitadores
76                 case ' ':
77                 case '\t':
78                     if (sb.length() > 0) {
79                         tokens.add(sb.toString());
80                         sb.setLength(0);
81                     }
82                     special = false;
83                     break;
84                 case ';':
85                     if (sb.length() > 0) {
86                         tokens.add(sb.toString());
87                         sb.setLength(0);
88                     }
89                     tokens.add(Character.toString(str[i]));
90                     special = false;
91                     break;
92                 //Cadena
93                 case '''':
94                     if (sb.length() > 0) {
95                         tokens.add(sb.toString());
96                         sb.setLength(0);
97                     }
98                     string = true;
99                     special = false;
100                    lastSpecial = 0;
101                    sb.append(str[i]);
102                    break;
103                //Tokens de un caracter
```

```
104         case ':':
105         case '(':
106         case ')':
107         case '{':
108         case '}':
109         case '[':
110         case ']':
111         case '*':
112         case ',':  
113             if (sb.length() > 0) {
114                 tokens.add(sb.toString());
115                 sb.setLength(0);
116             }
117             tokens.add(Character.toString(str[i]));
118             special = false;
119             break;
120             //Símbolos especiales
121         case '!':  
122             special = true;
123             if (sb.length() > 0) {
124                 tokens.add(sb.toString());
125                 sb.setLength(0);
126             }
127             sb.append(str[i]);
128             lastSpecial = str[i];
129             break;
130         case '=':  
131             if (!special) {
132                 special = true;
133                 if (sb.length() > 0) {
134                     tokens.add(sb.toString());
135                     sb.setLength(0);
136                 }
137                 sb.append(str[i]);
138                 lastSpecial = str[i];
```

```
139         } else {
140             special = false;
141             switch (lastSpecial) {
142                 case '=':
143                 case '>':
144                 case '<':
145                 case '!':
146                     sb.append(str[i]);
147                     tokens.add(sb.toString());
148                     sb.setLength(0);
149                     break;
150             default:
151                 if (sb.length() > 0) {
152                     tokens.add(sb.toString());
153                     sb.setLength(0);
154                 }
155                 sb.append(str[i]);
156                 break;
157             }
158             lastSpecial = 0;
159         }
160         break;
161     case '-':
162     case '>':
163         if (!special) {
164             special = true;
165             if (sb.length() > 0) {
166                 tokens.add(sb.toString());
167                 sb.setLength(0);
168             }
169             sb.append(str[i]);
170             lastSpecial = str[i];
171         } else {
172             special = false;
173             switch (lastSpecial) {
```

```
174         case '-':
175             sb.append(str[i]);
176             tokens.add(sb.toString());
177             sb.setLength(0);
178             break;
179         default:
180             if (sb.length() > 0) {
181                 tokens.add(sb.toString());
182                 sb.setLength(0);
183             }
184             sb.append(str[i]);
185             break;
186         }
187         lastSpecial = 0;
188     }
189     break;
190 case '<':
191     special = true;
192     if (sb.length() > 0) {
193         tokens.add(sb.toString());
194         sb.setLength(0);
195     }
196     sb.append(str[i]);
197     lastSpecial = str[i];
198     break;
199 case '+':
200     if (!special) {
201         special = true;
202         if (sb.length() > 0) {
203             tokens.add(sb.toString());
204             sb.setLength(0);
205         }
206         sb.append(str[i]);
207         lastSpecial = str[i];
208     } else {
```

```
209         special = false;
210         switch (lastSpecial) {
211             case '+':
212                 sb.append(str[i]);
213                 tokens.add(sb.toString());
214                 sb.setLength(0);
215                 break;
216             default:
217                 if (sb.length() > 0) {
218                     tokens.add(sb.toString());
219                     sb.setLength(0);
220                 }
221                 sb.append(str[i]);
222                 break;
223             }
224             lastSpecial = 0;
225         }
226         break;
227     case '/':
228         if (!special) {
229             special = true;
230             if (sb.length() > 0) {
231                 tokens.add(sb.toString());
232                 sb.setLength(0);
233             }
234             sb.append(str[i]);
235             lastSpecial = str[i];
236         } else {
237             special = false;
238             switch (lastSpecial) {
239                 case '/':
240                     sb.append(str[i]);
241                     tokens.add(sb.toString());
242                     sb.setLength(0);
243                     break;
```

```
244                     default:  
245                         if (sb.length() > 0) {  
246                             tokens.add(sb.toString());  
247                             sb.setLength(0);  
248                         }  
249                         sb.append(str[i]);  
250                         break;  
251                     }  
252                     lastSpecial = 0;  
253                 }  
254                 break;  
255             //Cualquier otro simbolo  
256             default:  
257                 if (special && sb.length() > 0) {  
258                     tokens.add(sb.toString());  
259                     sb.setLength(0);  
260                     special = false;  
261                     lastSpecial = 0;  
262                 }  
263                 sb.append(str[i]);  
264                 break;  
265             }  
266             if (i == str.length - 1 && sb.length() > 0) { // El ultimo caracter  
267                 tokens.add(sb.toString());  
268             }  
269         } else { //Se está leyendo una cadena  
270             sb.append(str[i]);  
271             if (str[i] == '"') { //Si la cadena termina, se completa el token  
272                 string = false;  
273                 tokens.add(sb.toString());  
274                 sb.setLength(0);  
275             }  
276             if (i == str.length - 1 && sb.length() > 0) { // El ultimo caracter  
277                 tokens.add(sb.toString());  
278             }  
279         }  
280     }  
281 }
```

```
279 }  
280 }  
281 }  
282     return true;  
283 }  
284 }
```

```
1 package itc.automatas2.lexico;
2
3 import itc.automatas2.estructuras.PilaErrores;
4 import itc.automatas2.estructuras.RegistroErr;
5 import itc.automatas2.estructuras.RegistroTS;
6 import itc.automatas2.estructuras.TablaSimbolos;
7 import itc.automatas2.lexico.automatas.*;
8 import itc.automatas2.misc.BaseErrores;
9 import itc.automatas2.misc.Error;
10
11 import java.io.FileNotFoundException;
12 import java.io.IOException;
13 import java.util.ArrayList;
14
15 public class AnalizadorLexico {
16     public TablaSimbolos tS;
17     private ArrayList<AutomataToken> automatas;
18     private AutomataReservadas automataReservadas;
19
20     /**
21      * Constructor de la clase
22      */
23     public AnalizadorLexico() {
24         automatas = new ArrayList<>();
25         automatas.add(new AutomataCadena());
26         automatas.add(new AutomataDecimal());
27         automatas.add(new AutomataNumero());
28         automatas.add(new AutomataIdentificador());
29         automataReservadas = new AutomataReservadas();
30     }
31
32     /**
33      * Analiza un archivo de código fuente.
34      *
35      * @param ruta La ruta del archivo.
```

```
36     * @return <code>true</code> si se aceptó el código, <code>false</code> si fue rechazado.
37     */
38     public boolean analizar(String ruta) {
39         TS = new TablaSimbolos();
40         String token;
41         int ultimoTipo = Tipos.VOID;
42         boolean foundVar = false;
43         boolean foundAssign = false;
44         boolean foundFunc = false;
45         boolean noValFound = false;
46         String lastIdentRef = null;
47         boolean error = false;
48         try {
49             Tokenizador tok = new Tokenizador(ruta);
50             while ((token = tok.siguienteToken()) != null) {
51                 // Bandera para saber si el token ya fue reconocido
52                 boolean found = false;
53
54                 // Si el token empieza como comentario se ignora por completo el resto de la línea
55                 if (token.startsWith("//") || token.startsWith("#")) {
56                     tok.siguienteLinea();
57                     continue;
58                 }
59
60                 if (!token.matches("[\u0020-\u007E]+")) { // Se verifica que el token contenga
61                     símbolos del alfabeto (ascii)
62                     PilaErrores.meter(new RegistroErr(120, tok.archivo.getNoLinea(), token)); // Si
63                     no, se crea un nuevo error
64                     error = true;
65                     continue;
66                 }
67
68                 // Se reconocen palabras reservadas primero
69                 if (automataReservadas.ejecutar(token)) {
70                     switch (token) {
```

```
69         case "if":
70             tS.meter(new RegistroTS(token, Tokens.T_IF, Tipos.VOID, tok.archivo.
71                                     getNoLinea()));
72             found = true;
73             break;
74         case "else":
75             tS.meter(new RegistroTS(token, Tokens.T_ELSE, Tipos.VOID, tok.archivo.
76                                     getNoLinea()));
77             found = true;
78             break;
79         case "while":
80             tS.meter(new RegistroTS(token, Tokens.T WHILE, Tipos.VOID, tok.archivo.
81                                     getNoLinea()));
82             found = true;
83             break;
84         case "for":
85             tS.meter(new RegistroTS(token, Tokens.T FOR, Tipos.VOID, tok.archivo.
86                                     getNoLinea()));
87             found = true;
88             break;
89         case "in":
90             tS.meter(new RegistroTS(token, Tokens.T_IN, Tipos.VOID, tok.archivo.
91                                     getNoLinea()));
92             found = true;
93             break;
94         case "func":
95             tS.meter(new RegistroTS(token, Tokens.T FUNC, Tipos.VOID, tok.archivo.
96                                     getNoLinea()));
97             foundFunc = true;
98             found = true;
99             break;
100            case "return":
101                tS.meter(new RegistroTS(token, Tokens.T RETURN, Tipos.VOID, tok.archivo.
102                                         .getNoLinea()));
103                found = true;
```

```
97                     break;
98
99         case "true":
100            tS.meter(new RegistroTS(token, Tokens.T_TRUE, Tipos.BOOL, tok.archivo.
101                getNoLinea())));
102
103            if (foundVar && foundAssign) {
104                tS.getByID(lastIdentRef).VAL = token;
105                foundVar = false;
106                foundAssign = false;
107            }
108            found = true;
109            break;
110
111         case "false":
112            tS.meter(new RegistroTS(token, Tokens.T_FALSE, Tipos.BOOL, tok.archivo.
113                getNoLinea())));
114
115            if (foundVar && foundAssign) {
116                tS.getByID(lastIdentRef).VAL = token;
117                foundVar = false;
118                foundAssign = false;
119            }
120            found = true;
121            break;
122
123         case "null":
124            tS.meter(new RegistroTS(token, Tokens.T_NULL, Tipos.VOID, tok.archivo.
125                getNoLinea())));
126
127            found = true;
128            break;
129
130         case "int":
131            tS.meter(new RegistroTS(token, Tokens.T_INT, Tipos.INT, tok.archivo.
132                getNoLinea())));
133
134            ultimoTipo = Tipos.INT;
135            foundFunc = false;
136            found = true;
137            break;
138
139         case "real":
140            tS.meter(new RegistroTS(token, Tokens.T_REAL, Tipos.REAL, tok.archivo.
```

```
127 getNoLinea());  
128         ultimoTipo = Tipos.REAL;  
129         foundFunc = false;  
130         found = true;  
131         break;  
132     case "bool":  
133         tS.meter(new RegistroTS(token, Tokens.T_BOOL, Tipos.BOOL, tok.archivo.  
getNoLinea()));  
134         ultimoTipo = Tipos.BOOL;  
135         foundFunc = false;  
136         found = true;  
137         break;  
138     case "string":  
139         tS.meter(new RegistroTS(token, Tokens.T_STRING, Tipos.STR, tok.archivo.  
getNoLinea()));  
140         ultimoTipo = Tipos.STR;  
141         foundFunc = false;  
142         found = true;  
143         break;  
144     case "print":  
145         tS.meter(new RegistroTS(token, Tokens.T_PRINT, Tipos.VOID, tok.archivo.  
getNoLinea()));  
146         found = true;  
147         break;  
148     case "println":  
149         tS.meter(new RegistroTS(token, Tokens.T_PRINTLN, Tipos.VOID, tok.  
archivo.getNoLinea()));  
150         found = true;  
151         break;  
152     case "read":  
153         tS.meter(new RegistroTS(token, Tokens.T_READ, Tipos.VOID, tok.archivo.  
getNoLinea()));  
154         found = true;  
155         break;  
156 }
```

```
157     }
158
159     //Símbolos
160     if (!found) {
161         switch (token) {
162             case ":":
163                 tS.meter(new RegistroTS(token, Tokens.T_COLON, Tipos.VOID, tok.archivo.
164                 getNoLinea()));
165                 found = true;
166                 break;
167             case ";":
168                 tS.meter(new RegistroTS(token, Tokens.T_SEMICOLON, Tipos.VOID, tok.
169                 archivo.getNoLinea()));
170                 found = true;
171                 break;
172             case "(":
173                 tS.meter(new RegistroTS(token, Tokens.T_LPAREN, Tipos.VOID, tok.archivo
174                 .getNoLinea()));
175                 found = true;
176                 break;
177             case ")":
178                 tS.meter(new RegistroTS(token, Tokens.T_RPAREN, Tipos.VOID, tok.archivo
179                 .getNoLinea()));
180                 found = true;
181                 break;
182             case "{":
183                 tS.meter(new RegistroTS(token, Tokens.T_LBRACE, Tipos.VOID, tok.archivo
184                 .getNoLinea()));
185                 found = true;
186                 break;
```

```
186         case "[":
187             tS.meter(new RegistroTS(token, Tokens.T_LBRACKET, Tipos.VOID, tok.
188             archivo.getNoLinea())));
189             found = true;
190             break;
191         case "]":
192             tS.meter(new RegistroTS(token, Tokens.T_RBRACKET, Tipos.VOID, tok.
193             archivo.getNoLinea())));
194             found = true;
195             break;
196         case "<=":
197             tS.meter(new RegistroTS(token, Tokens.T_LTE, Tipos.VOID, tok.archivo.
198             getNoLinea())));
199             found = true;
200             break;
201         case ">=":
202             tS.meter(new RegistroTS(token, Tokens.T_GTE, Tipos.VOID, tok.archivo.
203             getNoLinea())));
204             found = true;
205             break;
206         case "!=":
207             tS.meter(new RegistroTS(token, Tokens.T_NE, Tipos.VOID, tok.archivo.
208             getNoLinea())));
209             found = true;
210             break;
211         case "<":
212             tS.meter(new RegistroTS(token, Tokens.T_LT, Tipos.VOID, tok.archivo.
213             getNoLinea())));
214             found = true;
215             break;
216         case ">":
217             tS.meter(new RegistroTS(token, Tokens.T_GT, Tipos.VOID, tok.archivo.
218             getNoLinea())));
219             found = true;
220             break;
```

```
214         case "==" :
215             tS.meter(new RegistroTS(token, Tokens.T_EQ, Tipos.VOID, tok.archivo.
216                                     getNoLinea())));
217             found = true;
218             break;
219         case "=" :
220             tS.meter(new RegistroTS(token, Tokens.T_ASSIGN, Tipos.VOID, tok.archivo.
221                                     .getNoLinea())));
222             found = true;
223             break;
224         case "--" :
225             tS.meter(new RegistroTS(token, Tokens.T_DEC, Tipos.VOID, tok.archivo.
226                                     getNoLinea())));
227             found = true;
228             break;
229         case "++" :
230             tS.meter(new RegistroTS(token, Tokens.T_INC, Tipos.VOID, tok.archivo.
231                                     getNoLinea())));
232             found = true;
233             break;
234         case "!" :
235             tS.meter(new RegistroTS(token, Tokens.T_NOT, Tipos.VOID, tok.archivo.
236                                     getNoLinea())));
237             found = true;
238             break;
239         case "+" :
240             tS.meter(new RegistroTS(token, Tokens.T_PLUS, Tipos.VOID, tok.archivo.
241                                     getNoLinea())));
242             found = true;
243             break;
244         case "-" :
245             tS.meter(new RegistroTS(token, Tokens.T_MINUS, Tipos.VOID, tok.archivo.
246                                     getNoLinea())));
247             found = true;
248             break;
```

```
242             case "*":
243                 tS.meter(new RegistroTS(token, Tokens.T_STAR, Tipos.VOID, tok.archivo.
244                                         getNoLinea())));
245                 found = true;
246                 break;
247             case "/":
248                 tS.meter(new RegistroTS(token, Tokens.T_DIV, Tipos.VOID, tok.archivo.
249                                         getNoLinea())));
250                 found = true;
251                 break;
252             case ">":
253                 tS.meter(new RegistroTS(token, Tokens.T_RANGE, Tipos.VOID, tok.archivo.
254                                         getNoLinea())));
255                 found = true;
256                 break;
257             case ",":
258                 tS.meter(new RegistroTS(token, Tokens.T_COMMA, Tipos.VOID, tok.archivo.
259                                         getNoLinea())));
260                 found = true;
261                 break;
262             }
263             if (!foundAssign && foundVar && token.equals("=")) {
264                 foundAssign = true;
265             }
266             if (noValFound) {
267                 foundVar = false;
268                 foundAssign = false;
269             }
270             // Se prosigue con la evaluacion en los autómatas
271             if (!found) {
272                 for (AutomataToken a : automatas) {
```

```
273     if (!found && a.ejecutar(token)) {
274         RegistroTS reg;
275         switch (a.getTokenId()) {
276             // Identificador
277             case Tokens.T_VAR:
278                 if (!foundFunc) {
279                     if (!tS.contiene(token)) {
280                         reg = new RegistroTS(token, Tokens.T_VAR, ultimoTipo,
281                             tok.archivo.getNoLinea());
282                         switch (ultimoTipo) {
283                             case Tipos.INT:
284                                 reg.VAL = "0";
285                                 break;
286                             case Tipos.REAL:
287                                 reg.VAL = "0.0";
288                                 break;
289                             case Tipos.BOOL:
290                                 reg.VAL = "false";
291                                 break;
292                         }
293                         ultimoTipo = Tipos.VOID;
294                         foundVar = true;
295                         lastIdentRef = reg.ID;
296                         tS.meter(reg);
297                     } else {
298                         RegistroTS prev = tS.get(token);
299                         reg = new RegistroTS(token, prev.TOKEN_ID, Tipos.REF,
300                             tok.archivo.getNoLinea(), prev.ID);
301                         tS.meter(reg);
302                         lastIdentRef = prev.ID;
303                     }
304                 } else {
305                     tS.meter(new RegistroTS(token, Tokens.T_FUN, Tipos.VOID,
306                         tok.archivo.getNoLinea()));
307                     foundFunc = false;
308                 }
309             }
310         }
311     }
312 }
```

```
305
306
307
308
309
310
311     archivo.getNoLinea());
312
313
314
315
316
317
318
319
320
321     archivo.getNoLinea());
322
323
324
325
326
327
328
329
330
331     tok.archivo.getNoLinea());
332
333
334
335
336 }
```

}

found = true;

break;

// Constantes

case Tokens.T_STR_CONST:

reg = new RegistroTS(token, Tokens.T_STR_CONST, Tipos.STR, tok.

archivo.getNoLinea());

if (foundVar && foundAssign) {

tS.getByID(lastIdentRef).VAL = token.replaceAll("\\\"", "");

foundVar = false;

foundAssign = false;

}

tS.meter(reg);

found = true;

break;

case Tokens.T_INT_CONST:

reg = new RegistroTS(token, Tokens.T_INT_CONST, Tipos.INT, tok.

archivo.getNoLinea());

if (foundVar && foundAssign) {

tS.getByID(lastIdentRef).VAL = token;

foundVar = false;

foundAssign = false;

}

tS.meter(reg);

found = true;

break;

case Tokens.T_REAL_CONST:

reg = new RegistroTS(token, Tokens.T_REAL_CONST, Tipos.REAL,

tok.archivo.getNoLinea());

if (foundVar && foundAssign) {

tS.getByID(lastIdentRef).VAL = token;

foundVar = false;

foundAssign = false;

}

```

337                     ts.meter(reg);
338                     found = true;
339                     break;
340                 }
341             }
342         }
343     }
344     noValFound = !(foundVar || foundAssign);
345     // Si el token no fue reconocido, se marca error.
346     if (!found) {
347         PilaErrores.meter(new RegistroErr(110, tok.archivo.getNoLinea(), token));
348         error = true;
349     }
350 }
351 } catch (FileNotFoundException e) {
352     PilaErrores.meter(new RegistroErr(10, 0, ruta));
353     error = true;
354 } catch (IOException e) {
355     PilaErrores.meter(new RegistroErr(11, 0, ruta));
356     error = true;
357 }
358 return !error;
359 }
360 /**
361 * Imprime la tabla de símbolos en la salida estándar.
362 */
363 public void imprimirSimbolos() {
364     if (tS != null && tS.size() > 0) {
365         System.out.println(
366             "-----");
367         System.out.println(" | ");
368         System.out.println(" | ");

```

TABLA DE SÍMBOLOS

368

```
"-----"  
-----");  
369     System.out.printf("| %24s | %-30s | %-12s | %-12s | %24s |\n", "ID", "NOMBRE O LEXEMA",  
"TOKEN ID", "TIPO DE DATO", "VALOR");  
370     System.out.println(  
"-----"  
-----");  
371     for (RegistroTS reg : ts.registros()) {  
372         System.out.printf("| %24s | %-30s | %-12s | %-12s | %24s |\n", reg.ID, reg.NOMBRE,  
Tokens.nombres[reg.TOKEN_ID], Tipos.nombres[reg.TIPO], reg.VAL);  
373     }  
374     System.out.println(  
"-----"  
-----");  
375 } else {  
376     System.out.println("No se encontraron símbolos en el código");  
377 }  
378 }  
379 }
```

```
1 package itc.automatas2.lexico.automatas;
2
3 /**
4  * Clase prototipo para un autómata, que será extendida en cada autómata necesario.
5 */
6 public abstract class AutomataToken {
7     private final int TOKEN_ID;
8
9     AutomataToken(int TOKEN_ID) {
10         this.TOKEN_ID = TOKEN_ID;
11     }
12
13 /**
14     * Ejecuta el autómata de acuerdo al token dado.
15     *
16     * @param token Un <code>String</code> que contiene el token.
17     * @return El estado de aceptación <code>(true|false)</code>.
18     */
19     public abstract boolean ejecutar(String token);
20
21 /**
22     * Obtiene el ID del token que el autómata reconoce.
23     *
24     * @return El ID del token definido en {@link itc.automatas2.lexico.Tokens Tokens}.
25     */
26     public int getTokenId() {
27         return TOKEN_ID;
28     }
29 }
30 }
```

```
1 package itc.automatas2.lexico.automatas;
2
3 import itc.automatas2.lexico.Tokens;
4
5 /**
6  * Autómata que reconoce un identificador (variable).
7 */
8 public class AutomataCadena extends AutomataToken {
9
10    public AutomataCadena() {
11        super(Tokens.T_STR_CONST);
12    }
13
14    public boolean ejecutar(String token) {
15        return q0(0, token.toCharArray());
16    }
17
18    private boolean q0(int cont, char[] car) {
19        if (cont == car.length) {
20            return false;
21        } else {
22            if (Character.toString(car[cont]).equals("\\")) {
23                return this.q1(cont + 1, car);
24            }
25        }
26        return false;
27    }
28
29    private boolean q1(int cont, char[] car) {
30        if (cont == car.length) {
31            return false;
32        }
33        if (Character.toString(car[cont]).matches("[A-Za-z0-9{~-:@#/- ! ]")) {
34            return this.q1(cont + 1, car);
35        }
    }
```

```
36     if (Character.toString(car[cont]).equals("\\")) {
37         return this.q2(cont + 1, car);
38     }
39     return false;
40 }
41
42
43     private boolean q2(int cont, char[] car) {
44         return cont == car.length;
45     }
46 }
```

```
1 package itc.automatas2.lexico.automatas;
2
3 import itc.automatas2.lexico.Tokens;
4
5 /**
6  * Autómata que reconoce una constante numérica (entero).
7 */
8 public class AutomataNumero extends AutomataToken {
9
10    public AutomataNumero() {
11        super(Tokens.T_INT_CONST);
12    }
13
14    public boolean ejecutar(String token) {
15        return q0(0, token.toCharArray());
16    }
17
18    private boolean q0(int cont, char[] car) {
19        if (cont == car.length) {
20            return false;
21        } else {
22            if (Character.toString(car[cont]).matches("[0-9]")) {
23                return q1(cont + 1, car);
24
25            } else {
26                return false;
27            }
28
29        }
30    }
31
32    private boolean q1(int cont, char[] car) {
33        if (cont == car.length) {
34            return true;
35        }
36    }
37}
```

```
36     if (Character.toString(car[cont]).matches("[0-9]")) {  
37         return q1(cont + 1, car);  
38     } else {  
39         return false;  
40     }  
41 }  
42 }  
43 }  
44 }  
45 }
```

```
1 package itc.automatas2.lexico.automatas;
2
3 import itc.automatas2.lexico.Tokens;
4
5 /**
6  * Autómata que reconoce una constante decimal.
7 */
8 public class AutomataDecimal extends AutomataToken {
9
10    public AutomataDecimal() {
11        super(Tokens.T_REAL_CONST);
12    }
13
14    public boolean ejecutar(String token) {
15        return q0(0, token.toCharArray());
16    }
17
18    private boolean q0(int cont, char[] car) {
19        if (cont == car.length) {
20            return false;
21        }
22        if (Character.toString(car[cont]).matches("[0-9]")) {
23            return this.q1(cont + 1, car);
24        }
25        return false;
26    }
27
28    private boolean q1(int cont, char[] car) {
29        if (cont == car.length) {
30            return false;
31        }
32        if (Character.toString(car[cont]).matches("[0-9]")) {
33            return this.q1(cont + 1, car);
34        }
35        if (Character.toString(car[cont]).equals(".")) {
```

```
36         return this.q2(cont + 1, car);
37     }
38     return false;
39 }
40
41 private boolean q2(int cont, char[] car) {
42     if (cont == car.length) {
43         return false;
44     }
45     if (Character.toString(car[cont]).matches("[0-9]")) {
46         return this.q3(cont + 1, car);
47     }
48     return false;
49 }
50
51
52 private boolean q3(int cont, char[] car) {
53     if (cont == car.length) {
54         return true;
55     }
56     if (Character.toString(car[cont]).matches("[0-9]")) {
57         return this.q3(cont + 1, car);
58     }
59     return false;
60 }
61 }
62 }
```

```
1 package itc.automatas2.lexico.automatas;
2
3
4 public class AutomataReservadas {
5
6     public boolean ejecutar(String token) {
7         return q0(0, token.toCharArray());
8     }
9
10    private boolean q0(int cont, char[] car) {
11        if (cont == car.length) {
12            return false;
13        } else {
14            switch (Character.toString(car[cont])) {
15                case "w":
16                    return this.q1(cont + 1, car);
17                case "e":
18                    return this.q2(cont + 1, car);
19                case "i":
20                    return this.q3(cont + 1, car);
21                case "f":
22                    return this.q4(cont + 1, car);
23                case "r":
24                    return this.q5(cont + 1, car);
25                case "t":
26                    return this.q6(cont + 1, car);
27                case "n":
28                    return this.q7(cont + 1, car);
29                case "b":
30                    return this.q8(cont + 1, car);
31                case "s":
32                    return this.q9(cont + 1, car);
33                case "p":
34                    return this.q37(cont + 1, car);
35            }
40        }
41    }
42}
```

```
36         }
37     }
38     return false;
39 }
40
41 private boolean q1(int cont, char[] car) {
42     if (cont == car.length) {
43         return false;
44     } else {
45         if (Character.toString(car[cont]).equals("h")) {
46             return this.q10(cont + 1, car);
47         }
48     }
49     return false;
50 }
51
52 private boolean q10(int cont, char[] car) {
53     if (cont == car.length) {
54         return false;
55     } else {
56         if (Character.toString(car[cont]).equals("i")) {
57             return this.q11(cont + 1, car);
58         }
59     }
60     return false;
61 }
62
63 private boolean q11(int cont, char[] car) {
64     if (cont == car.length) {
65         return false;
66     } else {
67         if (Character.toString(car[cont]).equals("l")) {
68             return this.q12(cont + 1, car);
69         }
70     }
71 }
```

```
71         return false;
72     }
73
74     private boolean q12(int cont, char[] car) {
75         if (cont == car.length) {
76             return false;
77         } else {
78             if (Character.toString(car[cont]).equals("e")) {
79                 return this.qf(cont + 1, car);
80             }
81         }
82         return false;
83     }
84
85     private boolean q2(int cont, char[] car) {
86         if (cont == car.length) {
87             return false;
88         } else {
89             if (Character.toString(car[cont]).equals("l")) {
90                 return this.q13(cont + 1, car);
91             }
92         }
93         return false;
94     }
95
96     private boolean q13(int cont, char[] car) {
97         if (cont == car.length) {
98             return false;
99         } else {
100            if (Character.toString(car[cont]).equals("s")) {
101                return this.q14(cont + 1, car);
102            }
103        }
104        return false;
105    }
```

```
106
107     private boolean q14(int cont, char[] car) {
108         if (cont == car.length) {
109             return false;
110         } else {
111             if (Character.toString(car[cont]).equals("e")) {
112                 return this.qf(cont + 1, car);
113             }
114         }
115         return false;
116     }
117
118     private boolean q3(int cont, char[] car) {
119         if (cont == car.length) {
120             return false;
121         } else {
122             switch (Character.toString(car[cont])) {
123                 case "n":
124                     return this.q15(cont + 1, car);
125                 case "f":
126                     return this.qf(cont + 1, car);
127             }
128         }
129         return false;
130     }
131
132     private boolean q15(int cont, char[] car) {
133         if (cont == car.length) {
134             return true;
135         } else {
136             if (Character.toString(car[cont]).equals("t")) {
137                 return this.qf(cont + 1, car);
138             }
139         }
140         return false;
```

```
141     }
142
143     private boolean q4(int cont, char[] car) {
144         if (cont == car.length) {
145             return false;
146         } else {
147             switch (Character.toString(car[cont])) {
148                 case "o":
149                     return this.q16(cont + 1, car);
150                 case "u":
151                     return this.q17(cont + 1, car);
152                 case "a":
153                     return this.q19(cont + 1, car);
154             }
155         }
156         return false;
157     }
158
159     private boolean q16(int cont, char[] car) {
160         if (cont == car.length) {
161             return false;
162         } else {
163             if (Character.toString(car[cont]).equals("r")) {
164                 return this.qf(cont + 1, car);
165             }
166         }
167         return false;
168     }
169
170     private boolean q17(int cont, char[] car) {
171         if (cont == car.length) {
172             return false;
173         } else {
174             if (Character.toString(car[cont]).equals("n")) {
175                 return this.q18(cont + 1, car);
```

```
176         }
177     }
178     return false;
179 }
180
181     private boolean q18(int cont, char[] car) {
182         if (cont == car.length) {
183             return false;
184         } else {
185             if (Character.toString(car[cont]).equals("c")) {
186                 return this.qf(cont + 1, car);
187             }
188         }
189         return false;
190     }
191
192     private boolean q19(int cont, char[] car) {
193         if (cont == car.length) {
194             return false;
195         } else {
196             if (Character.toString(car[cont]).equals("l")) {
197                 return this.q20(cont + 1, car);
198             }
199         }
200         return false;
201     }
202
203     private boolean q20(int cont, char[] car) {
204         if (cont == car.length) {
205             return false;
206         } else {
207             if (Character.toString(car[cont]).equals("s")) {
208                 return this.q21(cont + 1, car);
209             }
210         }
211     }
```

```
211         return false;
212     }
213
214     private boolean q21(int cont, char[] car) {
215         if (cont == car.length) {
216             return false;
217         } else {
218             if (Character.toString(car[cont]).equals("e")) {
219                 return this.qf(cont + 1, car);
220             }
221         }
222         return false;
223     }
224
225     private boolean q5(int cont, char[] car) {
226         if (cont == car.length) {
227             return false;
228         } else {
229             if (Character.toString(car[cont]).equals("e")) {
230                 return this.q22(cont + 1, car);
231             }
232         }
233         return false;
234     }
235
236     private boolean q22(int cont, char[] car) {
237         if (cont == car.length) {
238             return false;
239         } else {
240             switch (Character.toString(car[cont])) {
241                 case "t":
242                     return this.q23(cont + 1, car);
243                 case "a":
244                     return this.q26(cont + 1, car);
245             }
246         }
247     }
```

```
246         }
247         return false;
248     }
249
250     private boolean q23(int cont, char[] car) {
251         if (cont == car.length) {
252             return false;
253         } else {
254             if (Character.toString(car[cont]).equals("u")) {
255                 return this.q24(cont + 1, car);
256             }
257         }
258         return false;
259     }
260
261     private boolean q24(int cont, char[] car) {
262         if (cont == car.length) {
263             return false;
264         } else {
265             if (Character.toString(car[cont]).equals("r")) {
266                 return this.q25(cont + 1, car);
267             }
268         }
269         return false;
270     }
271
272     private boolean q25(int cont, char[] car) {
273         if (cont == car.length) {
274             return false;
275         } else {
276             if (Character.toString(car[cont]).equals("n")) {
277                 return this.qf(cont + 1, car);
278             }
279         }
280         return false;
```

```
281     }
282
283     private boolean q26(int cont, char[] car) {
284         if (cont == car.length) {
285             return false;
286         } else {
287             switch (Character.toString(car[cont])) {
288                 case "l":
289                 case "d":
290                     return this.qf(cont + 1, car);
291                 }
292             }
293             return false;
294     }
295
296     private boolean q6(int cont, char[] car) {
297         if (cont == car.length) {
298             return false;
299         } else {
300             if (Character.toString(car[cont]).equals("r")) {
301                 return this.q27(cont + 1, car);
302             }
303         }
304         return false;
305     }
306
307     private boolean q27(int cont, char[] car) {
308         if (cont == car.length) {
309             return false;
310         } else {
311             if (Character.toString(car[cont]).equals("u")) {
312                 return this.q28(cont + 1, car);
313             }
314         }
315         return false;
```

```
316     }
317
318     private boolean q28(int cont, char[] car) {
319         if (cont == car.length) {
320             return false;
321         } else {
322             if (Character.toString(car[cont]).equals("e")) {
323                 return this.qf(cont + 1, car);
324             }
325         }
326         return false;
327     }
328
329     private boolean q7(int cont, char[] car) {
330         if (cont == car.length) {
331             return false;
332         } else {
333             if (Character.toString(car[cont]).equals("u")) {
334                 return this.q29(cont + 1, car);
335             }
336         }
337         return false;
338     }
339
340     private boolean q29(int cont, char[] car) {
341         if (cont == car.length) {
342             return false;
343         } else {
344             if (Character.toString(car[cont]).equals("l")) {
345                 return this.q30(cont + 1, car);
346             }
347         }
348         return false;
349     }
350 }
```

```
351     private boolean q30(int cont, char[] car) {
352         if (cont == car.length) {
353             return false;
354         } else {
355             if (Character.toString(car[cont]).equals("l")) {
356                 return this.qf(cont + 1, car);
357             }
358         }
359         return false;
360     }
361
362     private boolean q8(int cont, char[] car) {
363         if (cont == car.length) {
364             return false;
365         } else {
366             if (Character.toString(car[cont]).equals("o")) {
367                 return this.q31(cont + 1, car);
368             }
369         }
370         return false;
371     }
372
373     private boolean q31(int cont, char[] car) {
374         if (cont == car.length) {
375             return false;
376         } else {
377             if (Character.toString(car[cont]).equals("o")) {
378                 return this.q32(cont + 1, car);
379             }
380         }
381         return false;
382     }
383
384     private boolean q32(int cont, char[] car) {
385         if (cont == car.length) {
```

```
386         return false;
387     } else {
388         if (Character.toString(car[cont]).equals("l")) {
389             return this.qf(cont + 1, car);
390         }
391     }
392     return false;
393 }
394
395 private boolean q9(int cont, char[] car) {
396     if (cont == car.length) {
397         return false;
398     } else {
399         if (Character.toString(car[cont]).equals("t")) {
400             return this.q33(cont + 1, car);
401         }
402     }
403     return false;
404 }
405
406 private boolean q33(int cont, char[] car) {
407     if (cont == car.length) {
408         return false;
409     } else {
410         if (Character.toString(car[cont]).equals("r")) {
411             return this.q34(cont + 1, car);
412         }
413     }
414     return false;
415 }
416
417 private boolean q34(int cont, char[] car) {
418     if (cont == car.length) {
419         return false;
420     } else {
```

```
421         if (Character.toString(car[cont]).equals("i")) {
422             return this.q35(cont + 1, car);
423         }
424     }
425     return false;
426 }
427
428 private boolean q35(int cont, char[] car) {
429     if (cont == car.length) {
430         return false;
431     } else {
432         if (Character.toString(car[cont]).equals("n")) {
433             return this.q36(cont + 1, car);
434         }
435     }
436     return false;
437 }
438
439 private boolean q36(int cont, char[] car) {
440     if (cont == car.length) {
441         return false;
442     } else {
443         if (Character.toString(car[cont]).equals("g")) {
444             return this.qf(cont + 1, car);
445         }
446     }
447     return false;
448 }
449
450 private boolean q37(int cont, char[] car) {
451     if (cont == car.length) {
452         return false;
453     } else {
454         if (Character.toString(car[cont]).equals("r")) {
455             return this.q38(cont + 1, car);
```

```
456         }
457     }
458     return false;
459 }
460
461     private boolean q38(int cont, char[] car) {
462         if (cont == car.length) {
463             return false;
464         } else {
465             if (Character.toString(car[cont]).equals("i")) {
466                 return this.q39(cont + 1, car);
467             }
468         }
469         return false;
470     }
471
472     private boolean q39(int cont, char[] car) {
473         if (cont == car.length) {
474             return false;
475         } else {
476             if (Character.toString(car[cont]).equals("n")) {
477                 return this.q40(cont + 1, car);
478             }
479         }
480         return false;
481     }
482
483     private boolean q40(int cont, char[] car) {
484         if (cont == car.length) {
485             return false;
486         } else {
487             if (Character.toString(car[cont]).equals("t")) {
488                 return this.q41(cont + 1, car);
489             }
490         }
491     }
```

```
491         return false;
492     }
493
494     private boolean q41(int cont, char[] car) {
495         if (cont == car.length) {
496             return true;
497         } else {
498             if (Character.toString(car[cont]).equals("l")) {
499                 return this.q42(cont + 1, car);
500             }
501         }
502         return false;
503     }
504
505     private boolean q42(int cont, char[] car) {
506         if (cont == car.length) {
507             return false;
508         } else {
509             if (Character.toString(car[cont]).equals("n")) {
510                 return this.qf(cont + 1, car);
511             }
512         }
513         return false;
514     }
515
516     private boolean qf(int cont, char[] car) {
517         if (cont == car.length) {
518             return true;
519         } else {
520             return false;
521         }
522     }
523 }
524 }
```

```
1 package itc.automatas2.lexico.automatas;
2
3 import itc.automatas2.lexico.Tokens;
4
5 /**
6  * Autómata que reconoce un identificador (variable).
7 */
8 public class AutomataIdentificador extends AutomataToken {
9
10    public AutomataIdentificador() {
11        super(Tokens.T_VAR);
12    }
13
14    public boolean ejecutar(String token) {
15        return q0(0, token.toCharArray());
16    }
17
18    private boolean q0(int cont, char[] car) {
19        if (cont == car.length) {
20            return false;
21        } else {
22            if (Character.toString(car[cont]).matches("[A-Za-z_]")) {
23                return this.q1(cont + 1, car);
24            } else {
25                return false;
26            }
27        }
28    }
29
30    private boolean q1(int cont, char[] car) {
31        if (cont == car.length) {
32            return true;
33        }
34        if (Character.toString(car[cont]).matches("[A-Za-z0-9_]")) {
35            return this.q1(cont + 1, car);
36        }
37    }
38}
```

```
36         }
37         return false;
38
39     }
40
41 }
42
```

```
1 package itc.automatas2.sintactico;
2
3 /**
4  * Catálogo de ID's de las reglas de produccion.
5 */
6 public class ReglasProd {
7
8     //Reglas de produccion
9     public static final int R_BLOQUE = 100;
10    public static final int R_CALL = 101;
11    public static final int R_METODO = 102;
12    public static final int R_PARAMS = 103;
13    public static final int R_ARGS = 104;
14    public static final int R_RETORNO = 105;
15    public static final int R_IF = 106;
16    public static final int R_FOR = 107;
17    public static final int R_WHILE = 108;
18    public static final int R_LECTURA = 109;
19    public static final int R_IMPRESION = 110;
20    public static final int R_ASIGNACION = 111;
21    public static final int R_EXPR_REL = 112;
22    public static final int R_EXPR_ARITM = 113;
23
24    public static final String[] nombres = new String[]{
25        "R_BLOQUE",
26        "R_CALL",
27        "R_METODO",
28        "R_PARAMS",
29        "R_ARGS",
30        "R_RETORNO",
31        "R_IF",
32        "R_FOR",
33        "R_WHILE",
34        "R_LECTURA",
35        "R_IMPRESION",
```

```
36     "R_ASIGNACION",
37     "R_EXPR_REL",
38     "R_EXPR_ARITM"
39   } ;
40 }
```

```
1 package itc.automatas2.sintactico;
2
3 import itc.automatas2.estructuras.*;
4 import itc.automatas2.lexico.Tokens;
5
6
7 public class Estructuras {
8
9     public static int cont;
10
11    /**
12     * Analiza la estructura sintáctica del if y regresa su árbol
13     *
14     * @param i el índice del primer token en la tabla
15     * @param ts la tabla de símbolos
16     * @return el árbol sintáctico de la estructura
17     * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la
18     * secuencia.
19     * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
20     * estructura dentro de un bloque.
21     */
22    public static ArbolSintactico r_if(int i, TablaSimbolos ts) throws SecuenciaIncorrectaException,
23    EstructuraNoReconocidaException {
24        cont = i;
25        ArbolSintactico a = new ArbolSintactico();
26        RegistroTS reg;
27        NodoArbol actual;
28        actual = new NodoArbol();
29        actual.REGLA_ID = ReglasProd.R_IF;
30        actual.TOKEN_ID = Tokens.T_IF;
31        actual.REF = reg.ID;
32    }
```

```
33         a.meter(a.raiz, actual);

34

35     reg = ts.get(++cont);
36     if (reg.TOKEN_ID != Tokens.T_LPAREN) {
37         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
38         throw new SecuenciaIncorrectaException();
39     }
40     actual = new NodoArbol();
41     actual.TOKEN_ID = Tokens.T_LPAREN;
42     actual.REF = reg.ID;
43     a.meter(a.raiz, actual);

44

45     ArbolSintactico a2 = r_expr_rel(++cont, ts);
46     a.meter(a.raiz, a2.raiz);

47

48     reg = ts.get(cont);
49     if (reg.TOKEN_ID != Tokens.T_RPAREN) {
50         PilaErrores.meter(new RegistroErr(230, reg.LINE, reg.NOMBRE));
51         throw new SecuenciaIncorrectaException();
52     }
53     actual = new NodoArbol();
54     actual.TOKEN_ID = Tokens.T_RPAREN;
55     actual.REF = reg.ID;
56     a.meter(a.raiz, actual);

57

58     a2 = r_bloque(++cont, ts);
59     a.meter(a.raiz, a2.raiz);
60     reg = ts.get(cont);
61     if (reg != null) {
62         if (reg.TOKEN_ID != Tokens.T_ELSE) {
63             return a;
64         }
65         actual = new NodoArbol();
66         actual.TOKEN_ID = Tokens.T_ELSE;
67         actual.REF = reg.ID;
```

```
68         a.meter(a.raiz, actual);
69
70         cont++;
71         a2 = r_else(ts);
72         a.meter(a.raiz, a2.raiz);
73     }
74     return a;
75 }
76
77 /**
78 * Analiza la estructura sintáctica del else y regresa su árbol
79 *
80 * @param ts la tabla de símbolos
81 * @return el árbol sintáctico de la estructura
82 * @throws SecuenciaIncorrectaException      cuando se encuentra un token inesperado en la
secuencia.
83 * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
estructura dentro de un bloque.
84 */
85     private static ArbolSintactico r_else(TablaSimbolos ts) throws SecuenciaIncorrectaException,
EstructuraNoReconocidaException {
86         RegistroTS reg = ts.get(cont);
87         if (reg.TOKEN_ID == Tokens.T_IF) {
88             return r_if(cont, ts);
89         } else {
90             return r_bloque(cont, ts);
91         }
92     }
93
94 /**
95 * Analiza la estructura sintáctica del bloque y regresa su árbol
96 *
97 * @param i el índice del primer token en la tabla
98 * @param ts la tabla de símbolos
99 * @return el árbol sintáctico de la estructura
```

```
100     * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la  
secuencia.  
101    * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o  
estructura dentro de un bloque.  
102    */  
103    public static ArbolSintactico r_bloque(int i, TablaSimbolos ts) throws  
SecuenciaIncorrectaException, EstructuraNoReconocidaException {  
104        cont = i;  
105        ArbolSintactico a = new ArbolSintactico();  
106        RegistroTS reg;  
107        NodoArbol actual;  
108        actual = new NodoArbol();  
109        actual.REGLA_ID = ReglasProd.R_BLOQUE;  
110        a.raiz = actual;  
111  
112        reg = ts.get(cont);  
113        if (reg.TOKEN_ID != Tokens.T_LBRACE) {  
114            PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
115            throw new SecuenciaIncorrectaException();  
116        }  
117  
118        actual = new NodoArbol();  
119        actual.TOKEN_ID = Tokens.T_LBRACE;  
120        actual.REF = reg.ID;  
121        a.meter(a.raiz, actual);  
122        ArbolSintactico a2;  
123        cont++;  
124  
125        reg = ts.get(cont + 1);  
126        if (reg.TOKEN_ID == Tokens.T_RBRACE) {  
127            PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
128            throw new SecuenciaIncorrectaException();  
129        }  
130  
131        do {
```

```
132     switch (ObtenerReglaProd(cont, ts)) {
133         case ReglasProd.R_CALL:
134             a2 = r_call(cont, ts, false);
135             a.meter(a.raiz, a2.raiz);
136             break;
137         case ReglasProd.R_RETORNO:
138             a2 = r_retorno(cont, ts);
139             a.meter(a.raiz, a2.raiz);
140             break;
141         case ReglasProd.R_IF:
142             a2 = r_if(cont, ts);
143             a.meter(a.raiz, a2.raiz);
144             break;
145         case ReglasProd.R_FOR:
146             a2 = r_for(cont, ts);
147             a.meter(a.raiz, a2.raiz);
148             break;
149         case ReglasProd.R_WHILE:
150             a2 = r_while(cont, ts);
151             a.meter(a.raiz, a2.raiz);
152             break;
153         case ReglasProd.R_LECTURA:
154             a2 = r_lectura(cont, ts);
155             a.meter(a.raiz, a2.raiz);
156             break;
157         case ReglasProd.R_IMPRESION:
158             a2 = r_impresion(cont, ts);
159             a.meter(a.raiz, a2.raiz);
160             break;
161         case ReglasProd.R_ASIGNACION:
162             a2 = r_asignacion(cont, ts);
163             a.meter(a.raiz, a2.raiz);
164             break;
165         default:
166             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
```

```
167             throw new SecuenciaIncorrectaException();
168         }
169         reg = ts.get(cont);
170     } while (reg.TOKEN_ID != Tokens.T_RBRACE);
171
172     actual = new NodoArbol();
173     actual.TOKEN_ID = Tokens.T_RBRACE;
174     actual.REF = reg.ID;
175     a.meter(a.raiz, actual);
176
177     cont++;
178     return a;
179 }
180
181 /**
182 * Analiza la estructura sintáctica del call y regresa su árbol
183 *
184 * @param i el índice del primer token en la tabla
185 * @param ts la tabla de símbolos
186 * @return el árbol sintáctico de la estructura
187 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
188 */
189 public static ArbolSintactico r_call(int i, TablaSimbolos ts, boolean asignacion) throws
SecuenciaIncorrectaException {
190     cont = i;
191     ArbolSintactico a = new ArbolSintactico();
192     RegistroTS reg;
193     NodoArbol actual;
194     actual = new NodoArbol();
195     actual.REGLA_ID = ReglasProd.R_CALL;
196     a.raiz = actual;
197
198     reg = ts.get(cont);
199     actual = new NodoArbol();
200     actual.TOKEN_ID = Tokens.T_FUN;
```

```
201     actual.REF = reg.ID;
202     a.meter(a.raiz, actual);
203
204     reg = ts.get(++cont);
205     if (reg.TOKEN_ID != Tokens.T_LPAREN) {
206         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
207         throw new SecuenciaIncorrectaException();
208     }
209     actual = new NodoArbol();
210     actual.TOKEN_ID = Tokens.T_LPAREN;
211     actual.REF = reg.ID;
212     a.meter(a.raiz, actual);
213
214     reg = ts.get(++cont);
215     if (reg.TOKEN_ID != Tokens.T_RPAREN) {
216         ArbolSintactico a2 = r_params(cont, ts);
217         a.meter(a.raiz, a2.raiz);
218     }
219     reg = ts.get(cont);
220     if (reg.TOKEN_ID != Tokens.T_RPAREN) {
221         PilaErrores.meter(new RegistroErr(230, reg.LINE, reg.NOMBRE));
222         throw new SecuenciaIncorrectaException();
223     }
224     actual = new NodoArbol();
225     actual.TOKEN_ID = Tokens.T_RPAREN;
226     actual.REF = reg.ID;
227     a.meter(a.raiz, actual);
228
229     if (!asignacion) {
230         reg = ts.get(++cont);
231         if (reg.TOKEN_ID != Tokens.T_SEMICOLON) {
232             PilaErrores.meter(new RegistroErr(220, reg.LINE, reg.NOMBRE));
233             throw new SecuenciaIncorrectaException();
234         }
235         actual = new NodoArbol();
```

```
236         actual.TOKEN_ID = Tokens.T_SEMICOLON;
237         actual.REF = reg.ID;
238         a.meter(a.raiz, actual);
239     }
240     cont++;
241     return a;
242 }
243
244 /**
245 * Analiza la estructura sintáctica del metodo y regresa su árbol
246 *
247 * @param i el índice del primer token en la tabla
248 * @param ts la tabla de símbolos
249 * @return el árbol sintáctico de la estructura
250 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la
secuencia.
251 * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
estructura dentro de un bloque.
252 */
253 public static ArbolSintactico r_metodo(int i, TablaSimbolos ts) throws
SecuenciaIncorrectaException, EstructuraNoReconocidaException {
254     cont = i;
255     ArbolSintactico a = new ArbolSintactico();
256     RegistroTS reg;
257     NodoArbol actual;
258     actual = new NodoArbol();
259     actual.REGLA_ID = ReglasProd.R_METODO;
260     a.raiz = actual;
261
262     reg = ts.get(cont);
263     actual = new NodoArbol();
264     actual.TOKEN_ID = Tokens.T_FUNC;
265     actual.REF = reg.ID;
266     a.meter(a.raiz, actual);
267 }
```

```
268     reg = ts.get(++cont);
269     if (reg.TOKEN_ID != Tokens.T_FUN) {
270         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
271         throw new SecuenciaIncorrectaException();
272     }
273     actual = new NodoArbol();
274     actual.TOKEN_ID = Tokens.T_FUN;
275     actual.REF = reg.ID;
276     a.meter(a.raiz, actual);
277     RegistroTS funcName = reg;
278
279     reg = ts.get(++cont);
280     if (reg.TOKEN_ID != Tokens.T_LPAREN) {
281         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
282         throw new SecuenciaIncorrectaException();
283     }
284     actual = new NodoArbol();
285     actual.TOKEN_ID = Tokens.T_LPAREN;
286     actual.REF = reg.ID;
287     a.meter(a.raiz, actual);
288
289     reg = ts.get(++cont);
290     if (reg.TOKEN_ID != Tokens.T_RPAREN) {
291         ArbolesSintactico a2 = r_args(cont, ts);
292         a.meter(a.raiz, a2.raiz);
293     }
294     reg = ts.get(cont);
295     if (reg.TOKEN_ID != Tokens.T_RPAREN) {
296         PilaErrores.meter(new RegistroErr(230, reg.LINE, reg.NOMBRE));
297         throw new SecuenciaIncorrectaException();
298     }
299     actual = new NodoArbol();
300     actual.TOKEN_ID = Tokens.T_RPAREN;
301     actual.REF = reg.ID;
302     a.meter(a.raiz, actual);
```

```
303     reg = ts.get(++cont);
304     if (reg.TOKEN_ID == Tokens.T_COLON) {
305         actual = new NodoArbol();
306         actual.TOKEN_ID = Tokens.T_COLON;
307         actual.REF = reg.ID;
308         a.meter(a.raiz, actual);
309         reg = ts.get(++cont);
310         switch (reg.TOKEN_ID) {
311             case Tokens.T_INT:
312             case Tokens.T_REAL:
313             case Tokens.T_BOOL:
314             case Tokens.T_STRING:
315                 actual = new NodoArbol();
316                 actual.TOKEN_ID = reg.TOKEN_ID;
317                 actual.REF = reg.ID;
318                 a.meter(a.raiz, actual);
319                 funcName.TIPO = reg.TIPO;
320                 break;
321             default:
322                 PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
323                 throw new SecuenciaIncorrectaException();
324             }
325         }
326         cont++;
327     }
328     ArbolSintactico a2 = r_bloque(cont, ts);
329     a.meter(a.raiz, a2.raiz);
330     return a;
331 }
332 /**
333 * Analiza la estructura sintáctica del retorno y regresa su árbol
334 *
335 * @param i el índice del primer token en la tabla
336 * @param ts la tabla de símbolos
337 */
```

```
338     * @return el árbol sintáctico de la estructura
339     * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
340     */
341     public static ArbolSintactico r_retorno(int i, TablaSimbolos ts) throws
342         SecuenciaIncorrectaException {
343         cont = i;
344         ArbolSintactico a = new ArbolSintactico();
345         RegistroTS reg;
346         NodoArbol actual;
347         actual = new NodoArbol();
348         actual.REGLA_ID = ReglasProd.R_RETORNO;
349         a.raiz = actual;
350
351         reg = ts.get(cont);
352         actual = new NodoArbol();
353         actual.TOKEN_ID = Tokens.T_RETURN;
354         actual.REF = reg.ID;
355         a.meter(a.raiz, actual);
356
357         reg = ts.get(++cont);
358         switch (reg.TOKEN_ID) {
359             case Tokens.T_VAR:
360             case Tokens.T_INT_CONST:
361             case Tokens.T_REAL_CONST:
362             case Tokens.T_TRUE:
363             case Tokens.T_FALSE:
364             case Tokens.T_STR_CONST:
365                 actual = new NodoArbol();
366                 actual.TOKEN_ID = reg.TOKEN_ID;
367                 actual.REF = reg.ID;
368                 a.meter(a.raiz, actual);
369                 break;
370             default:
371                 PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
372                 throw new SecuenciaIncorrectaException();
```

```
372     }
373
374     reg = ts.get(++cont);
375     if (reg.TOKEN_ID != Tokens.T_SEMICOLON) {
376         PilaErrores.meter(new RegistroErr(220, reg.LINE, reg.NOMBRE));
377         throw new SecuenciaIncorrectaException();
378     }
379     actual = new NodoArbol();
380     actual.TOKEN_ID = Tokens.T_SEMICOLON;
381     actual.REF = reg.ID;
382     a.meter(a.raiz, actual);
383     cont++;
384     return a;
385 }
386
387 /**
388 * Analiza la estructura sintáctica del for y regresa su árbol
389 *
390 * @param i el índice del primer token en la tabla
391 * @param ts la tabla de símbolos
392 * @return el árbol sintáctico de la estructura
393 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la
secuencia.
394 * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
estructura dentro de un bloque.
395 */
396 public static ArbolSintactico r_for(int i, TablaSimbolos ts) throws
SecuenciaIncorrectaException, EstructuraNoReconocidaException {
397     cont = i;
398     ArbolSintactico a = new ArbolSintactico();
399     RegistroTS reg;
400     NodoArbol actual;
401     actual = new NodoArbol();
402     actual.REGLA_ID = ReglasProd.R_FOR;
403     a.raiz = actual;
```

```
404  
405     reg = ts.get(cont);  
406     actual = new NodoArbol();  
407     actual.TOKEN_ID = Tokens.T_FOR;  
408     actual.REF = reg.ID;  
409     a.meter(a.raiz, actual);  
410  
411     reg = ts.get(++cont);  
412     if (reg.TOKEN_ID != Tokens.T_LPAREN) {  
413         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
414         throw new SecuenciaIncorrectaException();  
415     }  
416     actual = new NodoArbol();  
417     actual.TOKEN_ID = Tokens.T_LPAREN;  
418     actual.REF = reg.ID;  
419     a.meter(a.raiz, actual);  
420  
421     reg = ts.get(++cont);  
422     if (reg.TOKEN_ID != Tokens.T_VAR) {  
423         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
424         throw new SecuenciaIncorrectaException();  
425     }  
426     actual = new NodoArbol();  
427     actual.TOKEN_ID = Tokens.T_VAR;  
428     actual.REF = reg.ID;  
429     a.meter(a.raiz, actual);  
430  
431     reg = ts.get(++cont);  
432     if (reg.TOKEN_ID != Tokens.T_IN) {  
433         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
434         throw new SecuenciaIncorrectaException();  
435     }  
436     actual = new NodoArbol();  
437     actual.TOKEN_ID = Tokens.T_IN;  
438     actual.REF = reg.ID;
```

```
439         a.meter(a.raiz, actual);  
440  
441         reg = ts.get(++cont);  
442         switch (reg.TOKEN_ID) {  
443             case Tokens.T_VAR:  
444             case Tokens.T_INT_CONST:  
445                 actual = new NodoArbol();  
446                 actual.TOKEN_ID = reg.TOKEN_ID;  
447                 actual.REF = reg.ID;  
448                 a.meter(a.raiz, actual);  
449                 break;  
450             default:  
451                 PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
452                 throw new SecuenciaIncorrectaException();  
453         }  
454  
455         reg = ts.get(++cont);  
456         if (reg.TOKEN_ID != Tokens.T_RANGE) {  
457             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
458             throw new SecuenciaIncorrectaException();  
459         }  
460         actual = new NodoArbol();  
461         actual.TOKEN_ID = Tokens.T_RANGE;  
462         actual.REF = reg.ID;  
463         a.meter(a.raiz, actual);  
464  
465         reg = ts.get(++cont);  
466         switch (reg.TOKEN_ID) {  
467             case Tokens.T_VAR:  
468             case Tokens.T_INT_CONST:  
469                 actual = new NodoArbol();  
470                 actual.TOKEN_ID = reg.TOKEN_ID;  
471                 actual.REF = reg.ID;  
472                 a.meter(a.raiz, actual);  
473                 break;
```

```
474     default:
475         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
476         throw new SecuenciaIncorrectaException();
477     }
478
479     reg = ts.get(++cont);
480     if (reg.TOKEN_ID != Tokens.T_RPAREN) {
481         PilaErrores.meter(new RegistroErr(230, reg.LINE, reg.NOMBRE));
482         throw new SecuenciaIncorrectaException();
483     }
484     actual = new NodoArbol();
485     actual.TOKEN_ID = Tokens.T_RPAREN;
486     actual.REF = reg.ID;
487     a.meter(a.raiz, actual);
488
489
490     Arbolesintactico a2 = r_bloque(++cont, ts);
491     a.meter(a.raiz, a2.raiz);
492     return a;
493 }
494
495 /**
496 * Analiza la estructura sintáctica del while y regresa su árbol
497 *
498 * @param i el índice del primer token en la tabla
499 * @param ts la tabla de símbolos
500 * @return el árbol sintáctico de la estructura
501 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la
secuencia.
502 * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
estructura dentro de un bloque.
503 */
504 public static Arbolesintactico r_while(int i, TablaSimbolos ts) throws
SecuenciaIncorrectaException, EstructuraNoReconocidaException {
505 }
```

```
506     cont = i;
507     ArbolSintactico a = new ArbolSintactico();
508     RegistroTS reg;
509     NodoArbol actual;
510     actual = new NodoArbol();
511     actual.REGLA_ID = ReglasProd.R WHILE;
512     a.raiz = actual;
513
514     reg = ts.get(cont);
515     actual = new NodoArbol();
516     actual.TOKEN_ID = Tokens.T WHILE;
517     actual.REF = reg.ID;
518     a.meter(a.raiz, actual);
519
520     reg = ts.get(++cont);
521     if (reg.TOKEN_ID != Tokens.T_LPAREN) {
522         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
523         throw new SecuenciaIncorrectaException();
524     }
525     actual = new NodoArbol();
526     actual.TOKEN_ID = Tokens.T_LPAREN;
527     actual.REF = reg.ID;
528     a.meter(a.raiz, actual);
529
530     ArbolSintactico a2 = r_expr_rel(++cont, ts);
531     a.meter(a.raiz, a2.raiz);
532
533     reg = ts.get(cont);
534     if (reg.TOKEN_ID != Tokens.T_RPAREN) {
535         PilaErrores.meter(new RegistroErr(230, reg.LINE, reg.NOMBRE));
536         throw new SecuenciaIncorrectaException();
537     }
538     actual = new NodoArbol();
539     actual.TOKEN_ID = Tokens.T_RPAREN;
540     actual.REF = reg.ID;
```

```
541         a.meter(a.raiz, actual);
542
543         a2 = r_bloque(++cont, ts);
544         a.meter(a.raiz, a2.raiz);
545
546         return a;
547     }
548
549
550     /**
551      * Analiza la estructura sintáctica de la lectura y regresa su árbol
552      *
553      * @param i el índice del primer token en la tabla
554      * @param ts la tabla de símbolos
555      * @return el árbol sintáctico de la estructura
556      * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
557      */
558     public static ArbolSintactico r_lec  
tura(int i, TablaSimbolos ts) throws
559     SecuenciaIncorrectaException {
560         cont = i;
561         ArbolSintactico a = new ArbolSintactico();
562         RegistroTS reg;
563         NodoArbol actual;
564         actual = new NodoArbol();
565         actual.REGLA_ID = ReglasProd.R_LECTURA;
566         a.raiz = actual;
567
568         reg = ts.get(cont);
569         actual = new NodoArbol();
570         actual.TOKEN_ID = Tokens.T_READ;
571         actual.REF = reg.ID;
572         a.meter(a.raiz, actual);
573
574         reg = ts.get(++cont);
575         if (reg.TOKEN_ID != Tokens.T_VAR) {
```

```
575         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
576         throw new SecuenciaIncorrectaException();
577     }
578     actual = new NodoArbol();
579     actual.TOKEN_ID = Tokens.T_VAR;
580     actual.REF = reg.ID;
581     a.meter(a.raiz, actual);
582
583     reg = ts.get(++cont);
584     if (reg.TOKEN_ID != Tokens.T_SEMICOLON) {
585         PilaErrores.meter(new RegistroErr(210, reg.LINE, reg.NOMBRE));
586         throw new SecuenciaIncorrectaException();
587     }
588     actual = new NodoArbol();
589     actual.TOKEN_ID = Tokens.T_SEMICOLON;
590     actual.REF = reg.ID;
591     a.meter(a.raiz, actual);
592     cont++;
593     return a;
594 }
595
596 /**
597 * Analiza la estructura sintáctica de la impresión y regresa su árbol
598 *
599 * @param i el índice del primer token en la tabla
600 * @param ts la tabla de símbolos
601 * @return el árbol sintáctico de la estructura
602 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
603 */
604 public static ArbolSintactico r_impresion(int i, TablaSimbolos ts) throws
SecuenciaIncorrectaException {
605     cont = i;
606     ArbolSintactico a = new ArbolSintactico();
607     RegistroTS reg;
608     NodoArbol actual;
```

```
609     actual = new NodoArbol();
610     actual.REGLA_ID = ReglasProd.R_IMPRESION;
611     a.raiz = actual;
612
613     reg = ts.get(cont);
614     actual = new NodoArbol();
615     actual.TOKEN_ID = reg.TOKEN_ID;
616     actual.REF = reg.ID;
617     a.meter(a.raiz, actual);
618
619
620     reg = ts.get(++cont);
621     switch (reg.TOKEN_ID) {
622         case Tokens.T_VAR:
623         case Tokens.T_STR_CONST:
624             actual = new NodoArbol();
625             actual.TOKEN_ID = reg.TOKEN_ID;
626             actual.REF = reg.ID;
627             a.meter(a.raiz, actual);
628             break;
629         default:
630             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
631             throw new SecuenciaIncorrectaException();
632     }
633
634     reg = ts.get(++cont);
635     if (reg.TOKEN_ID != Tokens.T_SEMICOLON) {
636         PilaErrores.meter(new RegistroErr(210, reg.LINE, reg.NOMBRE));
637         throw new SecuenciaIncorrectaException();
638     } else {
639         actual = new NodoArbol();
640         actual.TOKEN_ID = reg.TOKEN_ID;
641         actual.REF = reg.ID;
642         a.meter(a.raiz, actual);
643     }
```

```
644         cont++;
645         return a;
646     }
647
648     /**
649      * Analiza la estructura sintáctica de los parametros y regresa su árbol
650      *
651      * @param i el índice del primer token en la tabla
652      * @param ts la tabla de símbolos
653      * @return el árbol sintáctico de la estructura
654      * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
655      */
656     public static ArbolSintactico r_params(int i, TablaSimbolos ts) throws
657     SecuenciaIncorrectaException {
658         cont = i;
659         ArbolSintactico a = new ArbolSintactico();
660         RegistroTS reg;
661         NodoArbol actual;
662         actual = new NodoArbol();
663         actual.REGLA_ID = ReglasProd.R_PARAMS;
664         a.raiz = actual;
665
666         do {
667             reg = ts.get(cont);
668             switch (reg.TOKEN_ID) {
669                 case Tokens.T_VAR:
670                 case Tokens.T_INT_CONST:
671                 case Tokens.T_REAL_CONST:
672                 case Tokens.T_TRUE:
673                 case Tokens.T_FALSE:
674                 case Tokens.T_STR_CONST:
675                     actual = new NodoArbol();
676                     actual.TOKEN_ID = reg.TOKEN_ID;
677                     actual.REF = reg.ID;
678                     a.meter(a.raiz, actual);
```

```
678         break;
679     default:
680         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
681         throw new SecuenciaIncorrectaException();
682     }
683
684     reg = ts.get(++cont);
685     if (reg.TOKEN_ID == Tokens.T_COMMA) {
686         actual = new NodoArbol();
687         actual.TOKEN_ID = Tokens.T_COMMA;
688         actual.REF = reg.ID;
689         a.meter(a.raiz, actual);
690         cont++;
691     }
692     } while (reg.TOKEN_ID == Tokens.T_COMMA);
693     return a;
694 }
695 /**
696 * Analiza la estructura sintáctica de los argumentos y regresa su árbol
697 *
698 * @param i el índice del primer token en la tabla
699 * @param ts la tabla de símbolos
700 * @return el árbol sintáctico de la estructura
701 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
702 */
703 public static ArbolSintactico r_args(int i, TablaSimbolos ts) throws
704     SecuenciaIncorrectaException {
705     cont = i;
706     ArbolSintactico a = new ArbolSintactico();
707     RegistroTS reg;
708     NodoArbol actual;
709     actual = new NodoArbol();
710     actual.REGLA_ID = ReglasProd.R_ARGS;
711     a.raiz = actual;
```

```
712
713     do {
714         reg = ts.get(cont);
715         switch (reg.TOKEN_ID) {
716             case Tokens.T_INT:
717             case Tokens.T_REAL:
718             case Tokens.T_BOOL:
719             case Tokens.T_STRING:
720                 actual = new NodoArbol();
721                 actual.TOKEN_ID = reg.TOKEN_ID;
722                 actual.REF = reg.ID;
723                 a.meter(a.raiz, actual);
724                 break;
725             default:
726                 PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
727                 throw new SecuenciaIncorrectaException();
728         }
729
730         reg = ts.get(++cont);
731         if (reg.TOKEN_ID != Tokens.T_VAR) {
732             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
733             throw new SecuenciaIncorrectaException();
734         } else {
735             actual = new NodoArbol();
736             actual.TOKEN_ID = Tokens.T_VAR;
737             actual.REF = reg.ID;
738             a.meter(a.raiz, actual);
739         }
740
741         reg = ts.get(++cont);
742         if (reg.TOKEN_ID == Tokens.T_COMMA) {
743             actual = new NodoArbol();
744             actual.TOKEN_ID = Tokens.T_COMMA;
745             actual.REF = reg.ID;
746             a.meter(a.raiz, actual);
```

```
747             cont++;
748         }
749     } while (reg.TOKEN_ID == Tokens.T_COMMA);
750     return a;
751 }
752 }
753 /**
754 * Analiza la estructura sintáctica de la asignacion y regresa su árbol
755 *
756 * @param i el índice del primer token en la tabla
757 * @param ts la tabla de símbolos
758 * @return el árbol sintáctico de la estructura
759 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
760 */
761 public static ArbolSintactico r_asignacion(int i, TablaSimbolos ts) throws
762 SecuenciaIncorrectaException {
763     cont = i;
764     ArbolSintactico a = new ArbolSintactico();
765     RegistroTS reg;
766     NodoArbol actual;
767     actual = new NodoArbol();
768     actual.REGLA_ID = ReglasProd.R_ASIGNACION;
769     a.raiz = actual;
770
771     reg = ts.get(cont);
772
773     switch (reg.TOKEN_ID) {
774         case Tokens.T_INT:
775         case Tokens.T_REAL:
776         case Tokens.T_BOOL:
777         case Tokens.T_STRING:
778             actual = new NodoArbol();
779             actual.TOKEN_ID = reg.TOKEN_ID;
780             actual.REF = reg.ID;
```

```
781         a.meter(a.raiz, actual);
782         reg = ts.get(++cont);
783         break;
784     }
785
786     if (reg.TOKEN_ID != Tokens.T_VAR) {
787         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
788         throw new SecuenciaIncorrectaException();
789     } else {
790         actual = new NodoArbol();
791         actual.TOKEN_ID = Tokens.T_VAR;
792         actual.REF = reg.ID;
793         a.meter(a.raiz, actual);
794     }
795
796     reg = ts.get(++cont);
797
798     if (reg.TOKEN_ID == Tokens.T_SEMICOLON) {
799         actual = new NodoArbol();
800         actual.TOKEN_ID = Tokens.T_SEMICOLON;
801         actual.REF = reg.ID;
802         a.meter(a.raiz, actual);
803         cont++;
804         return a;
805     } else if (reg.TOKEN_ID != Tokens.T_ASSIGN) {
806         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
807         throw new SecuenciaIncorrectaException();
808     } else {
809         actual = new NodoArbol();
810         actual.TOKEN_ID = Tokens.T_ASSIGN;
811         actual.REF = reg.ID;
812         a.meter(a.raiz, actual);
813     }
814
815     cont++;
```

```
816     reg = ts.get(cont + 1);
817     ArbolSintactico a2;
818     switch (reg.TOKEN_ID) {
819         case Tokens.T_PLUS:
820         case Tokens.T_MINUS:
821         case Tokens.T_STAR:
822         case Tokens.T_DIV:
823             a2 = r_expr_aritm(cont, ts);
824             a.meter(a.raiz, a2.raiz);
825             break;
826         case Tokens.T_EQ:
827         case Tokens.T_LT:
828         case Tokens.T_GT:
829         case Tokens.T_NE:
830         case Tokens.T_LTE:
831         case Tokens.T_GTE:
832             a2 = r_expr_rel(cont, ts);
833             a.meter(a.raiz, a2.raiz);
834             break;
835     default:
836         reg = ts.get(cont);
837         switch (reg.TOKEN_ID) {
838             case Tokens.T_INT_CONST:
839             case Tokens.T_REAL_CONST:
840             case Tokens.T_STR_CONST:
841             case Tokens.T_TRUE:
842             case Tokens.T_FALSE:
843             case Tokens.T_VAR:
844                 actual = new NodoArbol();
845                 actual.TOKEN_ID = reg.TOKEN_ID;
846                 actual.REF = reg.ID;
847                 a.meter(a.raiz, actual);
848                 cont++;
849                 break;
850         case Tokens.T_FUN:
```

```
851             a2 = r_call(cont, ts, true);
852             a.meter(a.raiz, a2.raiz);
853             break;
854         }
855     }
856     reg = ts.get(cont);
857     if (reg.TOKEN_ID != Tokens.T_SEMICOLON) {
858         PilaErrores.meter(new RegistroErr(210, reg.LINE, reg.NOMBRE));
859         throw new SecuenciaIncorrectaException();
860     }
861     actual = new NodoArbol();
862     actual.TOKEN_ID = Tokens.T_SEMICOLON;
863     actual.REF = reg.ID;
864     a.meter(a.raiz, actual);
865     cont++;
866     return a;
867 }
868 /**
869 * Analiza la estructura sintáctica de la expresión relacional y regresa su árbol
870 *
871 * @param i el índice del primer token en la tabla
872 * @param ts la tabla de símbolos
873 * @return el árbol sintáctico de la estructura
874 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la
875 * secuencia.
876 * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
877 * estructura dentro de un bloque.
878 */
879 public static ArbolSintactico r_expr_rel(int i, TablaSimbolos ts) throws
880 SecuenciaIncorrectaException {
881     cont = i;
882     ArbolSintactico a = new ArbolSintactico();
883     RegistroTS reg;
```

```
883     NodoArbol actual;
884     actual = new NodoArbol();
885     actual.REGLA_ID = ReglasProd.R_EXPR_REL;
886     a.raiz = actual;
887
888     reg = ts.get(cont);
889
890     switch (reg.TOKEN_ID) {
891         case Tokens.T_INT_CONST:
892         case Tokens.T_REAL_CONST:
893         case Tokens.T_TRUE:
894         case Tokens.T_FALSE:
895         case Tokens.T_VAR:
896         case Tokens.T_STR_CONST:
897             actual = new NodoArbol();
898             actual.TOKEN_ID = reg.TOKEN_ID;
899             actual.REF = reg.ID;
900             a.meter(a.raiz, actual);
901             break;
902         default:
903             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
904             throw new SecuenciaIncorrectaException();
905     }
906     reg = ts.get(++cont);
907
908     switch (reg.TOKEN_ID) {
909         case Tokens.T_GTE:
910         case Tokens.T_LTE:
911         case Tokens.T_GT:
912         case Tokens.T_LT:
913         case Tokens.T_EQ:
914         case Tokens.T_NE:
915             actual = new NodoArbol();
916             actual.TOKEN_ID = reg.TOKEN_ID;
917             actual.REF = reg.ID;
```

```
918         a.meter(a.raiz, actual);
919         break;
920     default:
921         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
922         throw new SecuenciaIncorrectaException();
923     }
924     reg = ts.get(++cont);
925
926     switch (reg.TOKEN_ID) {
927         case Tokens.T_INT_CONST:
928         case Tokens.T_REAL_CONST:
929         case Tokens.T_TRUE:
930         case Tokens.T_FALSE:
931         case Tokens.T_VAR:
932         case Tokens.T_STR_CONST:
933             actual = new NodoArbol();
934             actual.TOKEN_ID = reg.TOKEN_ID;
935             actual.REF = reg.ID;
936             a.meter(a.raiz, actual);
937             break;
938         default:
939             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
940             throw new SecuenciaIncorrectaException();
941     }
942     cont++;
943     return a;
944 }
945
946 /**
947 * Analiza la estructura sintáctica de la expresión aritmética y regresa su árbol
948 *
949 * @param i el índice del primer token en la tabla
950 * @param ts la tabla de símbolos
951 * @return el árbol sintáctico de la estructura
952 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
```

```
953     */
954     public static ArbolSintactico r_expr_aritm(int i, TablaSimbolos ts) throws
955         SecuenciaIncorrectaException {
956         cont = i;
957         ArbolSintactico a = new ArbolSintactico();
958         RegistroTS reg;
959         NodoArbol actual;
960         actual = new NodoArbol();
961         actual.REGLA_ID = ReglasProd.R_EXPR_ARITM;
962         a.raiz = actual;
963
964         reg = ts.get(cont);
965
966         switch (reg.TOKEN_ID) {
967             case Tokens.T_INT_CONST:
968             case Tokens.T_REAL_CONST:
969             case Tokens.T_TRUE:
970             case Tokens.T_FALSE:
971             case Tokens.T_VAR:
972             case Tokens.T_STR_CONST:
973                 actual = new NodoArbol();
974                 actual.TOKEN_ID = reg.TOKEN_ID;
975                 actual.REF = reg.ID;
976                 a.meter(a.raiz, actual);
977                 break;
978             default:
979                 PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
980                 throw new SecuenciaIncorrectaException();
981         }
982         reg = ts.get(++cont);
983
984         switch (reg.TOKEN_ID) {
985             case Tokens.T_PLUS:
986             case Tokens.T_MINUS:
987             case Tokens.T_STAR:
```

```
987     case Tokens.T_DIV:
988         actual = new NodoArbol();
989         actual.TOKEN_ID = reg.TOKEN_ID;
990         actual.REF = reg.ID;
991         a.meter(a.raiz, actual);
992         break;
993     default:
994         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
995         throw new SecuenciaIncorrectaException();
996     }
997     reg = ts.get(++cont);
998
999     switch (reg.TOKEN_ID) {
1000         case Tokens.T_INT_CONST:
1001         case Tokens.T_REAL_CONST:
1002         case Tokens.T_TRUE:
1003         case Tokens.T_FALSE:
1004         case Tokens.T_VAR:
1005         case Tokens.T_STR_CONST:
1006             actual = new NodoArbol();
1007             actual.TOKEN_ID = reg.TOKEN_ID;
1008             actual.REF = reg.ID;
1009             a.meter(a.raiz, actual);
1010             break;
1011         default:
1012             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
1013             throw new SecuenciaIncorrectaException();
1014     }
1015     cont++;
1016     return a;
1017 }
1018
1019
1020 /**
1021 * Obtiene la regla de produccion a la que pertenece, con base en el primer token
```

```
1022     *
1023     * @param i el índice del primer token en la tabla
1024     * @param ts la tabla de símbolos
1025     * @return el árbol sintáctico de la estructura
1026     * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la
1027     * secuencia.
1028     */
1029     public static int ObtenerReglaProd(int i, TablaSimbolos ts) throws
1030         EstructuraNoReconocidaException {
1031         RegistroTS reg;
1032         cont = i;
1033         reg = ts.get(cont);
1034
1035         switch (reg.TOKEN_ID) {
1036             case Tokens.T_FUN:
1037                 return ReglasProd.R_CALL;
1038             case Tokens.T_FUNC:
1039                 return ReglasProd.R_METODO;
1040             case Tokens.T_RETURN:
1041                 return ReglasProd.R_RETORNO;
1042             case Tokens.T_IF:
1043                 return ReglasProd.R_IF;
1044             case Tokens.T_FOR:
1045                 return ReglasProd.R_FOR;
1046             case Tokens.T_WHILE:
1047                 return ReglasProd.R WHILE;
1048             case Tokens.T_READ:
1049                 return ReglasProd.R LECTURA;
1050             case Tokens.T_PRINT:
1051             case Tokens.T_PRINTLN:
1052                 return ReglasProd.R_IMPRESION;
1053             case Tokens.T_VAR:
1054             case Tokens.T_REAL:
```

```
1054         case Tokens.T_INT:  
1055         case Tokens.T_BOOL:  
1056         case Tokens.T_STRING:  
1057             return ReglasProd.R_ASIGNACION;  
1058     }  
1059     throw new EstructuraNoReconocidaException();  
1060 }  
1061 }
```

```
1 package itc.automatas2.sintactico;
2
3 import itc.automatas2.estructuras.*;
4 import itc.automatas2.lexico.Tokens;
5 import itc.automatas2.misc.BaseErrores;
6 import itc.automatas2.misc.Error;
7
8 import java.util.ArrayList;
9
10
11 public class AnalizadorSintactico {
12
13     public ArrayList<ArbolSintactico> arboles;
14
15     /**
16      * Ejecuta el análisis sintáctico del programa a partir de su tabla de símbolos.
17      *
18      * @param ts la tabla de símbolos.
19      * @return <code>true</code> si el análisis no generó errores.
20      */
21     public boolean analizarTablaSimbolos (TablaSimbolos ts) {
22         arboles = new ArrayList<>();
23         Estructuras.cont = 0;
24         try {
25             do {
26                 switch (Estructuras.ObtenerReglaProd(Estructuras.cont, ts)) {
27                     case ReglasProd.R_CALL:
28                         arboles.add(
29                             Estructuras.r_call(Estructuras.cont, ts, false)
30                         );
31                         break;
32                     case ReglasProd.R_METODO:
33                         arboles.add(
34                             Estructuras.r_metodo(Estructuras.cont, ts)
35                         );
36                 }
37             }
38             catch (Exception e) {
39                 System.out.println("Error en la ejecución del análisis sintáctico: " + e.getMessage());
40             }
41         }
42         catch (Exception e) {
43             System.out.println("Error en la ejecución del análisis sintáctico: " + e.getMessage());
44         }
45     }
46 }
```

```
36             break;
37         case ReglasProd.R_RETORNO:
38             arboles.add(
39                 Estructuras.r_retorno(Estructuras.cont, ts)
40             );
41             break;
42         case ReglasProd.R_IF:
43             arboles.add(
44                 Estructuras.r_if(Estructuras.cont, ts)
45             );
46             break;
47         case ReglasProd.R_FOR:
48             arboles.add(
49                 Estructuras.r_for(Estructuras.cont, ts)
50             );
51             break;
52         case ReglasProd.R_WHILE:
53             arboles.add(
54                 Estructuras.r_while(Estructuras.cont, ts)
55             );
56             break;
57         case ReglasProd.R_LECTURA:
58             arboles.add(
59                 Estructuras.r_lectura(Estructuras.cont, ts)
60             );
61             break;
62         case ReglasProd.R_IMPRESION:
63             arboles.add(
64                 Estructuras.r_impresion(Estructuras.cont, ts)
65             );
66             break;
67         case ReglasProd.R_ASIGNACION:
68             arboles.add(
69                 Estructuras.r_asignacion(Estructuras.cont, ts)
70             );

```

```
71                     break;
72
73                 }
74             } while (Estructuras.cont < ts.size());
75         } catch (EstructuraNoReconocidaException | SecuenciaIncorrectaException e) {
76             arboles = null;
77             return false;
78         } catch (NullPointerException e) {
79             PilaErrores.meter(new RegistroErr(12, -1, ""));
80             arboles = null;
81             return false;
82         }
83         return true;
84     }
85
86 /**
87 * Imprime los arboles generados despues de la validacion de la secuencia de los tokens
88 */
89 public void imprimirArboles() {
90     if (arboles != null)
91         for (ArbolSintactico arbol : arboles) {
92             System.out.println(arbol.toString());
93             System.out.println();
94         }
95     }
96
97 public boolean tieneArboles() {
98     return arboles != null && arboles.size() > 0;
99 }
100 }
```

```
1 package itc.automatas2.sintactico;
2
3 public class SecuenciaIncorrectaException extends Exception {
4
5     public SecuenciaIncorrectaException() {
6         super();
7     }
8
9     public SecuenciaIncorrectaException(String message) {
10        super(message);
11    }
12
13 }
```

```
1 package itc.automatas2.sintactico;
2
3 public class EstructuraNoReconocidaException extends Exception {
4
5     public EstructuraNoReconocidaException() {
6         super();
7     }
8
9     public EstructuraNoReconocidaException(String message) {
10        super(message);
11    }
12
13 }
```

```
1 package itc.automatas2.estructuras;
2
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileNotFoundException;
6 import java.io.FileReader;
7 import java.io.IOException;
8
9 /**
10  * Clase para lectura de un archivo de texto. Se puede leer línea por línea o por completo.
11 */
12 public class Archivo {
13     private final String RUTA;
14     private final File file;
15     private BufferedReader reader;
16     private int line;
17
18     /**
19      * Crea un objeto de la clase.
20      *
21      * @param ruta La ruta del archivo.
22      * @throws FileNotFoundException Si el archivo no existe.
23      */
24     public Archivo(String ruta) throws FileNotFoundException {
25         this.RUTA = ruta;
26         this.file = new File(ruta);
27         //Si el archivo no existe tira excepción, que otra clase se encargue
28         if (!file.exists())
29             throw new FileNotFoundException();
30         this.reader = new BufferedReader(new FileReader(file));
31         this.line = 0;
32     }
33
34     /**
35      * Lee la siguiente línea del archivo.
```

```
36     *
37     * @return La siguiente línea, <code>null</code> si se llegó al final del archivo. Una vez que se
38     * llega al final del archivo, el mismo se cierra.
39     * @throws IOException Si ocurre un error en la lectura o el archivo ya se cerró.
40     */
41     public String leerLinea() throws IOException {
42         String tmp = reader.readLine();
43         line++;
44         if (tmp == null)
45             reader.close();
46         return tmp;
47     }
48
49     /**
50      * Obtiene el número de línea actual.
51      *
52      * @return El número de línea actual.
53      */
54     public int getNoLinea() {
55         return line;
56     }
57 }
```

```
1 package itc.automatas2.estructuras;
2
3 import itc.automatas2.lexico.Tokens;
4 import itc.automatas2.sintactico.ReglasProd;
5
6 import java.util.ArrayList;
7
8 /**
9  * Clase nodo que forma parte del árbol sintáctico.
10 */
11 public class NodoArbol {
12     public NodoArbol padre;
13     public ArrayList<NodoArbol> hijos;
14     public String REF;
15     public int TOKEN_ID;
16     public int REGLA_ID;
17
18     public NodoArbol() {
19         hijos = new ArrayList<>();
20     }
21
22     @Override
23     public String toString() {
24         String s = TOKEN_ID != 0 ? Tokens.nombres[TOKEN_ID] : "<" + ReglasProd.nombres[REGLA_ID - 100
25 ] + ">";
26         s += REF != null && !REF.isEmpty() ? ":" + REF : "";
27         return s;
28     }
29 }
```

```
1 package itc.automatas2.estructuras;
2
3 /**
4  * Registro de la tabla de símbolos, que contiene el nombre, el ID del token, el tipo, el valor y el
5  * número de linea.
6 */
7 public class RegistroTS {
8     public final String ID;
9     public final String NOMBRE;
10    public final int TOKEN_ID;
11    public int TIPO;
12    public String VAL;
13    public final int LINE;
14
15    /**
16     * Constructor de la clase
17     *
18     * @param NOMBRE
19     * @param TOKEN_ID
20     * @param TIPO
21     * @param LINE
22     */
23    public RegistroTS(String NOMBRE, int TOKEN_ID, int TIPO, int LINE) {
24        this.ID = TablaSimbolos.newID(NOMBRE);
25        this.NOMBRE = NOMBRE;
26        this.TOKEN_ID = TOKEN_ID;
27        this.TIPO = TIPO;
28        this.VAL = null;
29        this.LINE = LINE;
30    }
31
32    /**
33     * Constructor de la clase
34     *
35     * @param NOMBRE
```

```
35     * @param TOKEN_ID
36     * @param TIPO
37     * @param LINE
38     * @param VAL
39     */
40     public RegistroTS(String NOMBRE, int TOKEN_ID, int TIPO, int LINE, String VAL) {
41         this.ID = TablaSimbolos.newID(NOMBRE);
42         this.NOMBRE = NOMBRE;
43         this.TOKEN_ID = TOKEN_ID;
44         this.TIPO = TIPO;
45         this.VAL = VAL;
46         this.LINE = LINE;
47     }
48 }
49
```

```
1 package itc.automatas2.estructuras;
2
3 import itc.automatas2.lexico.Tokens;
4 import itc.automatas2.misc.BaseErrores;
5 import itc.automatas2.misc.Error;
6
7 import java.util.ArrayDeque;
8
9 /**
10 *
11 */
12 public class PilaErrores {
13     private static ArrayDeque<RegistroErr> pila = new ArrayDeque<>();
14
15     /**
16      * Inserta un nuevo error a la pila
17      *
18      * @param error Un objeto del tipo {@link RegistroErr RegistroErr}
19      */
20     public static void meter(RegistroErr error) {
21         pila.push(error);
22     }
23
24     /**
25      * Remueve el último objeto de la pila
26      *
27      * @return Un objeto del tipo {@link RegistroErr RegistroErr}
28      */
29     public static RegistroErr sacar() {
30         return pila.pop();
31     }
32
33     /**
34      * Obtiene el tamaño de la pila
35      *
```

```
36     * @return Un entero que representa el tamaño de la pila
37     */
38     public static int size() {
39         return pila.size();
40     }
41
42     public static void imprimirErrores() {
43         if (pila.size() > 0)
44             while (PilaErrores.size() > 0) {
45                 RegistroErr reg = PilaErrores.sacar();
46                 Error err = BaseErrores.getError(reg.ERR_ID);
47                 switch (reg.ERR_ID) {
48                     //De sistema
49                     case 10:
50                     case 11:
51                         err.setReason(String.format("(RUTA: %s)", reg.LEXEMA));
52                         break;
53                     case 12:
54                         err.setReason("Verifique que todas las llaves estén cerradas o que la última
sentencia termine en '\";\".'");
55                         break;
56                     //Léxicos
57                     case 110:
58                         err.setReason(String.format("No se pudo identificar el token \"%s\" en la
línea %d", reg.LEXEMA, reg.LINEA_N));
59                         break;
60                     case 120:
61                         err.setReason(String.format("El token '%s' en la línea %d contiene símbolos
no reconocibles (fuera del código ASCII). ", reg.LEXEMA, reg.LINEA_N));
62                         break;
63                     //Sintacticos
64                     case 210:
65                     case 220:
66                     case 230:
67                     case 250:
```

```
68         err.setReason(String.format("(línea %d)", reg.LINEA_N));
69         break;
70     case 240:
71         err.setReason(String.format("No se esperaba el token %s ('%s') cerca de la
línea %d.", Tokens.nombres[reg.TOKEN_ID], reg.LEXEMA, reg.LINEA_N));
72         break;
73     //Semánticos
74     case 310:
75     case 320:
76     case 330:
77     case 340:
78     case 350:
79     case 360:
80     case 370:
81         err.setReason(reg.LEXEMA);
82         break;
83     }
84     System.err.println(err);
85 }
86 }
87 }
```

```
1 package itc.automatas2.estructuras;
2
3 /**
4  * Registro de la pila de errores, que contiene el ID del error, el número de linea, y el lexema que
5  * lo causó.
6 */
7 public class RegistroErr {
8     public final int ERR_ID;
9     public final int LINEA_N;
10    public final String LEXEMA;
11    public int TOKEN_ID;
12
13    /**
14     * Constructor de la clase
15     *
16     * @param ERR_ID
17     * @param LINEA_N
18     * @param LEXEMA
19     */
20    public RegistroErr(int ERR_ID, int LINEA_N, String LEXEMA) {
21        this.ERR_ID = ERR_ID;
22        this.LINEA_N = LINEA_N;
23        this.LEXEMA = LEXEMA;
24    }
25
26    /**
27     * Constructor de la clase
28     *
29     * @param ERR_ID
30     * @param LINEA_N
31     * @param LEXEMA
32     * @param TOKEN_ID
33     */
34    public RegistroErr(int ERR_ID, int LINEA_N, String LEXEMA, int TOKEN_ID) {
35        this.ERR_ID = ERR_ID;
```

```
35         this.LINEA_N = LINEA_N;  
36         this.LEXEMA = LEXEMA;  
37         this.TOKEN_ID = TOKEN_ID;  
38     }  
39 }  
40
```

```
1 package itc.automatas2.estructuras;
2
3 import java.util.ArrayDeque;
4
5 /**
6  * Clase que representa una tabla de símbolos.
7 */
8 public class TablaSimbolos {
9     private ArrayDeque<RegistroTS> tabla;
10
11    /**
12     * Constructor de la clase
13     */
14    public TablaSimbolos() {
15        tabla = new ArrayDeque<>();
16    }
17
18    /**
19     * Metodo para meter un nuevo registro a la tabla
20     *
21     * @param registro Un objeto del tipo {@link RegistroTS RegistroTS}.
22     */
23    public void meter(RegistroTS registro) {
24        tabla.add(registro);
25    }
26
27    /**
28     * Metodo para verificar la existencia de un registro en la tabla
29     *
30     * @param llave La llave del registro.
31     * @return <code>true</code> si el registro existe, <code>false</code> si no.
32     */
33    public boolean contiene(String llave) {
34        return tabla.stream().anyMatch(reg -> reg.NOMBRE.equals(llave));
35    }
}
```

```
36
37     /**
38      * Metodo para obtener los valores de los registro en la tabla
39      *
40      * @return Un objeto del tipo {@link RegistroTS RegistroTS}.
41      */
42     public ArrayDeque<RegistroTS> registros() {
43         return tabla.clone();
44     }
45
46     /**
47      * Obtiene el tamaño de la tabla
48      *
49      * @return Un entero que representa el tamaño de la tabla
50      */
51     public int size() {
52         return tabla.size();
53     }
54
55     /**
56      * Obtiene un registro por su nombre o lexema
57      *
58      * @param nombre el nombre o lexema
59      * @return el registro encontrado
60      */
61     public RegistroTS get(String nombre) {
62         return tabla.stream().filter(reg -> nombre.equals(reg.NOMBRE)).findFirst().get();
63     }
64
65     /**
66      * Obtiene un registro por su índice
67      *
68      * @param i el índice
69      * @return el registro encontrado
70      */
```

```
71     public RegistroTS get(int i) {
72         int j = 0;
73         for (RegistroTS reg : tabla) {
74             if (j != i) {
75                 j++;
76             } else return reg;
77         }
78         return null;
79     }
80
81     /**
82      * Obtiene un registro por su ID
83      *
84      * @param id el ID
85      * @return el registro encontrado
86      */
87     public RegistroTS getByID(String id) {
88         return tabla.stream().filter(reg -> id.equals(reg.ID)).findFirst().get();
89     }
90
91     /**
92      * Regresa una cadena única como ID.
93      *
94      * @param token el token para obtener una cadena
95      * @return una cadena de ID única
96      */
97     static String newID(String token) {
98         return String.format("%x%08x", System.nanoTime(), Math.abs(token.hashCode()));
99     }
100 }
```

```
1 package itc.automatas2.estructuras;
2
3 /**
4  * Árbol sintáctico para usarse en el análisis.
5 */
6 public class ArbolSintactico {
7     public NodoArbol raiz;
8
9     /**
10      * Agrega un nodo a la lista de hijos de otro
11      *
12      * @param padre el {@link NodoArbol} padre.
13      * @param hijo el {@link NodoArbol} hijo.
14      */
15     public void meter(NodoArbol padre, NodoArbol hijo) {
16         padre.hijos.add(hijo);
17         hijo.padre = padre;
18     }
19
20     public String toString() {
21         return toString(raiz, 0);
22     }
23
24     private String toString(NodoArbol nodo, int niv) {
25         StringBuilder sb = new StringBuilder();
26         if (niv > 0) {
27             sb.append(" | ");
28             for (int i = 0; i < niv; i++) {
29                 sb.append("   ");
30             }
31             sb.deleteCharAt(sb.length() - 1);
32         }
33         sb.append(" |--");
34         sb.append(nodo.toString());
35         sb.append("\n");
```

```
36     if (nodo.hijos.size() > 0) {  
37         for (NodoArbol hijo : nodo.hijos) {  
38             sb.append(toString(hijo, niv + 1));  
39         }  
40     }  
41     return sb.toString();  
42 }  
43 }  
44 }
```

Anexo

Documentación complementaria

Propuesta de árbol sintáctico

Para la creación de la estructura del árbol primero se realizó una investigación previa para estar documentados en el uso del árbol dentro del compilador, y así poder identificar cómo sería más conveniente el diseño de la estructura.

Inicialmente se debatió la diferencia entre manejar los datos dentro de una pila para cada estructura generada, pero con base en la investigación el equipo concluyó que el árbol sería la mejor opción. De este modo, es posible seguir la secuencia en la que se generaron las sentencias según el recorrido más conveniente en el proceso.

En el árbol, se ingresan los tokens correspondientes a las estructuras una vez se valida que es correcto sintácticamente, de acuerdo al orden establecido en la definición del lenguaje.

Cuando se detecta que es una estructura dentro de otra estructura (como el caso de un bloque dentro de una condicional), se ingresa un nodo con el nombre de la regla de producción que identifica a la estructura anidada, y a partir de ese nodo se ingresan los nodos correspondientes a sus tokens como sus hijos.

De esta manera se genera un árbol por cada estructura sintáctica inicial, y una cantidad indefinida de subniveles según el número de estructuras anidadas que se contengan.

La estructura de cada nodo está definida en la sección de estructuras del documento de la actividad 3.

Impresión de los árboles sintácticos

Cuando se analizó el proceso del análisis sintáctico, se llegó a la problemática de cómo demostrar que los árboles fueron generados de manera exitosa. Para ello, el equipo se planteó el reto de mostrar de manera gráfica los árboles generados por el analizador sintáctico.

Como consideración, es necesario llevar la relación del nivel actual para saber el tamaño de la indentación que se tomará en cada impresión de un nodo, esto para llevar un formato donde se puedan apreciar descendientes de sus nodos padre.

Dibujar un árbol con gráficos en pantalla dentro de un contenedor sería bastante complicado, en especial para árboles muy complejos como los que puede generar el análisis sintáctico. Además, dentro del alcance del presente proyecto no se estableció el dibujo de los árboles como prioridad principal. Debido a esto, y a nuestras necesidades, se decidió mostrar los árboles en forma de texto a través de la salida estándar.

Lo que se muestra en la salida es una representación simplificada del árbol inspirada en el comando *tree* de MS-DOS, en donde cada nodo que conforma el árbol se muestra en su respectivo nivel, teniendo en cuenta su nodo padre (y por ende el nodo raíz de la estructura). De este modo, todos los tokens que conforman la estructura son representados mostrando el Token ID y el ID único en la tabla de símbolos.

Dependiendo de los atributos del nodo, su representación cambia. Si el nodo identifica a una regla de producción, se representa como el nombre de la misma entre *menor que* y *mayor que* (ejemplo: "<R_IF>"). Si no, el nodo identifica a un token, y se representa sólo con los atributos mencionados en el párrafo anterior.

Así, se pueden apreciar los tokens que forman parte de la estructura, y con la ayuda de la tabla de símbolos se pueden identificar con facilidad sus atributos, como su valor o ubicación en el código.

La siguiente imagen, es una captura del árbol que se muestra en el analizador sintáctico.



```
ANALISIS SINTACTICO DEL PROGRAMA C:\USERS\ANDRE\Desktop\asd.e1
El analizador sintactico declaró el código como válido
Árboles construidos:
|--<R_ASIGNACION>
|  |--T_INT: 994fc50c0c2000197ef
|  |--T_VAR: 994fc8d635000000078
|  |--T_ASSIGN: 994fc8faf70000003d
|  |--T_INT_CONST: 994fc9189d000000032
|  |--T_SEMICOLON: 994fcb8eb670000003b
```

Implementación de resaltado de sintaxis

Durante el desarrollo de la interfaz gráfica, se nos hizo la sugerencia de añadir resaltado de sintaxis al editor de texto.

Inicialmente, se consideraron diferentes posibilidades. Debido a que la interfaz de nuestro proyecto está basada en componentes de *Swing*, siendo el contenedor del texto un *JTextArea*, una solución viable es la extensión de sus clases auxiliares encargadas de mostrar el texto dentro del componente de modo que ciertas palabras clave sean resaltadas con colores a elegir.

Tomando esto en cuenta, el tiempo requerido para implementar este comportamiento puede ser bastante extenso, ya que requiere la comprensión total de los componentes de *Swing*. Por esta razón, se optó por implementar una librería que se ajustara a las necesidades del proyecto.

Al final, nuestra decisión fue implementar la librería [RSyntaxTextArea](#). Esta es desarrollada por el usuario *bobbylight*, y puede ser hallada en su repositorio de GitHub (<https://git.io/vprA5>). El repositorio contiene una *wiki* que, si bien no es extensa, contiene los detalles necesarios para su implementación.

Una vez añadida la librería al *classpath* del proyecto, sus componentes se pueden usar como cualquier componente de *Swing*. Si así se desea, se puede añadir a la biblioteca de componentes del diseñador de un IDE como IntelliJ o NetBeans, lo cual fue de gran ayuda para agilizar el desarrollo de la interfaz.

La librería contiene dos componentes que consideramos como el núcleo de la misma, *RTextScrollPane* y *RSyntaxTextArea*. El primero es una extensión de *JScrollPane* que permite mostrar números de línea además de implementar *code folding*, mientras que el segundo es el contenedor de texto principal que se encarga de resaltar la sintaxis. Detalles específicos de su integración en el proyecto se pueden encontrar en el repositorio de la librería.

Cabe mencionar que, además de contar con soporte para algunos de los lenguajes de programación más comunes, se puede añadir soporte para un nuevo lenguaje. La forma más conveniente de hacerlo es mediante JFlex (<=1.4.1), creando la definición de los tokens del lenguaje de acuerdo a los requerimientos de la librería.

En el repositorio se encuentran las definiciones de todos los lenguajes que soporta la librería en formato *flex* (<https://git.io/vprAc>). Como equipo, recomendamos tomar la definición del lenguaje más parecido al que se desee añadir soporte, modificarla de acuerdo a la de este, e integrar la misma con la librería. Este proceso se documenta con mayor detalle en la *wiki* del repositorio (<https://git.io/vprAn>). Más adelante, se anexa la especificación de nuestro lenguaje prototípico en formato *flex*.

Finalmente, los aspectos visuales del editor pueden configurarse desde el diseñador del IDE. Para más información se puede consultar el *javadoc* de la librería misma (<https://goo.gl/gMb4JQ>).

Anexo

Definición del lenguaje
en JFlex

```
1 package itc.automatas2.gui.lib;
2
3 import java.io.*;
4 import javax.swing.text.Segment;
5
6 import org.fife.ui.rsyntaxtextarea.*;
7
8 %%
9
10 %public
11 %class E1TokenMaker
12 %extends AbstractJFlexCTokenMaker
13 %unicode
14 %type org.fife.ui.rsyntaxtextarea.Token
15
16
17 %}
18
19
20 /**
21 * Constructor. This must be here because JFlex does not generate a
22 * no-parameter constructor.
23 */
24 public E1TokenMaker() {
25     super();
26 }
27
28
29 /**
30 * Adds the token specified to the current linked list of tokens.
31 *
32 * @param tokenType The token's type.
33 * @see #addToken(int, int, int)
34 */
35 private void addHyperlinkToken(int start, int end, int tokenType) {
```

```
36         int so = start + offsetShift;
37         addToken(zzBuffer, start,end, tokenType, so, true);
38     }
39
40
41     /**
42      * Adds the token specified to the current linked list of tokens.
43      *
44      * @param tokenType The token's type.
45      */
46     private void addToken(int tokenType) {
47         addToken(zzStartRead, zzMarkedPos-1, tokenType);
48     }
49
50
51     /**
52      * Adds the token specified to the current linked list of tokens.
53      *
54      * @param tokenType The token's type.
55      */
56     private void addToken(int start, int end, int tokenType) {
57         int so = start + offsetShift;
58         addToken(zzBuffer, start,end, tokenType, so);
59     }
60
61
62     /**
63      * Adds the token specified to the current linked list of tokens.
64      *
65      * @param array The character array.
66      * @param start The starting offset in the array.
67      * @param end The ending offset in the array.
68      * @param tokenType The token's type.
69      * @param startOffset The offset in the document at which this token
70      *                   occurs.
```

```
71     */
72     @Override
73     public void addToken(char[] array, int start, int end, int tokenType, int startOffset) {
74         super.addToken(array, start, end, tokenType, startOffset);
75         zzStartRead = zzMarkedPos;
76     }
77
78
79 /**
80 * {@inheritDoc}
81 */
82 @Override
83 public String[] getLineCommentStartAndEnd(int languageIndex) {
84     return new String[] { "//", null };
85 }
86
87
88 /**
89 * Returns the first token in the linked list of tokens generated
90 * from <code>text</code>. This method must be implemented by
91 * subclasses so they can correctly implement syntax highlighting.
92 *
93 * @param text The text from which to get tokens.
94 * @param initialTokenType The token type we should start with.
95 * @param startOffset The offset into the document at which
96 *        <code>text</code> starts.
97 * @return The first <code>Token</code> in a linked list representing
98 *        the syntax highlighted text.
99 */
100 public Token getTokenList(Segment text, int initialTokenType, int startOffset) {
101
102     resetTokenList();
103     this.offsetShift = -text.offset + startOffset;
104
105     // Start off in the proper state.
```

```
106     int state = Token.NULL;
107     switch (initialTokenType) {
108         default:
109             state = Token.NULL;
110     }
111
112     s = text;
113     try {
114         yyreset(zzReader);
115         yybegin(state);
116         return yylex();
117     } catch (IOException ioe) {
118         ioe.printStackTrace();
119         return new TokenImpl();
120     }
121
122 }
123
124
125 /**
126 * Refills the input buffer.
127 *
128 * @return      <code>true</code> if EOF was reached, otherwise
129 *              <code>false</code>.
130 */
131 private boolean zzRefill() {
132     return zzCurrentPos>=s.offset+s.count;
133 }
134
135
136 /**
137 * Resets the scanner to read from a new input stream.
138 * Does not close the old reader.
139 *
140 * All internal variables are reset, the old input stream
```

```
141     * <b>cannot</b> be reused (internal buffer is discarded and lost).  
142     * Lexical state is set to <tt>YY_INITIAL</tt>.   
143     *  
144     * @param reader    the new input stream  
145     */  
146     public final void yyreset(Reader reader) {  
147         // 's' has been updated.  
148         zzBuffer = s.array;  
149         /*  
150             * We replaced the line below with the two below it because zzRefill  
151             * no longer "refills" the buffer (since the way we do it, it's always  
152             * "full" the first time through, since it points to the segment's  
153             * array). So, we assign zzEndRead here.  
154         */  
155         //zzStartRead = zzEndRead = s.offset;  
156         zzStartRead = s.offset;  
157         zzEndRead = zzStartRead + s.count - 1;  
158         zzCurrentPos = zzMarkedPos = s.offset;  
159         zzLexicalState = YYINITIAL;  
160         zzReader = reader;  
161         zzAtBOL = true;  
162         zzAtEOF = false;  
163     }  
164  
165  
166 %}  
167  
168  
169 /* C1.1 - Line terminators. */  
170 NewlineCharacter           = ([\n])  
171  
172 /* C.1.2 - Whitespace. */  
173 Whitespace                 = ([\t ]+)  
174  
175 /* C.1.3 - Comments */
```

```

176 InputCharacter = ([^\n])
177 InputCharacters = ({InputCharacter}+)
178
179 SingleLineComment = ((// | #) ([^/]{InputCharacters}?) ?)
180
181
182 /* C.1.5 - Unicode character escape sequences. */
183 UnicodeEscape1 = ("\\u" {HexDigit} {HexDigit} {HexDigit} {HexDigit})
184 UnicodeEscape2 = ("\\U" {HexDigit} {HexDigit} {HexDigit} {HexDigit} {
    HexDigit} {HexDigit} {HexDigit})
185 UnicodeEscapeSequence = ({UnicodeEscape1} | {UnicodeEscape2})
186
187 /* C1.6 - Identifiers. */
188 LetterCharacter = ([A-Za-z]) /* Not accurate - many more Unicode letters,
    Unicode escapes */
189 /*
190 CombiningCharacter = ()
191 */
192 DecimalDigitCharacter = ([0-9])
193 ConnectingCharacter = ([_\\$])
194 /*
195 FormattingCharacter = ()
196 */
197 /*
198 IdentifierPartCharacter = ({LetterCharacter} | {DecimalDigitCharacter} | {
    ConnectingCharacter} | {CombiningCharacter} | {FormattingCharacter})
199 */
200 IdentifierPartCharacter = ({LetterCharacter} | {DecimalDigitCharacter} | {
    ConnectingCharacter})
201 IdentifierPartCharacters = ({IdentifierPartCharacter}+)
202 IdentifierStartCharacter = ({LetterCharacter} | [_\\$])
203 IdentifierOrKeyword = ({IdentifierStartCharacter}{IdentifierPartCharacters}?)?
204 Identifier = ("@"?{IdentifierOrKeyword})
205 /* NOTE: The two below aren't from the C# spec, but we add them so we can */
206 /* highlight errors. */

```

```

207 NonSeparator      = (((^\\t\\f\\r\\n\\ \\()\\{\\}\\[\\]\\;\\,.\\.=\\>\\<\\!\\~\\?\\:\\+\\-\\*\\/\\&\\|\\^\\%\\\"\\'])|"#"|"\\"))
208 ErrorIdentifier          = ({NonSeparator}+)
209
210 /* C1.8 - Literals. */
211 BooleanLiteral           = ("true"|"false")
212 DecimalDigit             = ([0-9])
213 DecimalDigits            = ({DecimalDigit}+)
214 IntegerTypeSuffix        = ([[uU][lL]?)|([lL][uU]?))
215 DecimalIntegerLiteral    = ({DecimalDigits}{IntegerTypeSuffix}?)
216 HexDigit                 = ([0-9A-Fa-f])
217 HexDigits                = ({HexDigit}+)
218 HexadecimalIntegerLiteral= ("0"[xX]{HexDigits}{IntegerTypeSuffix}?)|([+\\-])
219 Sign                      = ([+\\-])
220 ExponentPart             = ([eE]{Sign}?(DecimalDigits))
221 RealTypeSuffix            = ([fFdDmM])
222 RealHelper1               = ({DecimalDigits}."."){DecimalDigits}{ExponentPart}?
223 RealHelper2               = ("."{DecimalDigits}{ExponentPart}?{RealTypeSuffix}?)
224 RealHelper3               = ({DecimalDigits}{ExponentPart}{RealTypeSuffix}?)
225 RealHelper4               = ({DecimalDigits}{RealTypeSuffix})
226 RealLiteral               = ({RealHelper1}|{RealHelper2}|{RealHelper3}|{RealHelper4})
227 ErrorNumberFormat         = (((DecimalIntegerLiteral)|{HexadecimalIntegerLiteral})|{RealLiteral}){NonSeparator}+
228
229 SimpleEscapeSequence      = ("\\\\"[\\'\\\"\\\\0abfnrtv])
230 HexadecimalEscapeSequence= ("\\\\x"[HexDigit]{HexDigit}?(HexDigit)?{HexDigit}?)|
231
232
233 SingleRegularStringLiteralCharacter= ([^\\\"\\\\\\n])
234 RegularStringLiteralCharacter= ({SingleRegularStringLiteralCharacter}|{SimpleEscapeSequence}|{HexadecimalEscapeSequence}|{UnicodeEscapeSequence})
235 RegularStringLiteralCharacters= ({RegularStringLiteralCharacter}+)
236 RegularStringLiteral       = ([\\"]{RegularStringLiteralCharacters}?[\\"])
237 UnclosedRegularStringLiteral= ([\\"]([\\\\.|[^\\"\\"])*[^\\"]?)|
238 ErrorRegularStringLiteral = ({UnclosedRegularStringLiteral}[\\"])

```

```

239
240 /* C.1.9 - Operators and Punctuators. */
241 OOPHelper1 = (":")
242 OOPHelper2 = ("+" | "-" | "*" | "/" | "!")
243 OOPHelper3 = ("==" | "<" | ">" | "++" | "--")
244 OOPHelper4 = ("===" | "!=" | "<=" | ">=" | "+=" | "-=" | "*=" | "/=" | "**=")
245 OperatorOrPunctuator = ({OOPHelper1} | {OOPHelper2} | {OOPHelper3} | {OOPHelper4})
246 /* NOTE: We distinguish between operators and separators (punctuators), but */
247 /* the C# spec doesn't, so the stuff below isn't in the spec. */
248 Separator = ([\{\}\[\]\(\)])
249 Separator2 = ([,;])
250
251 %%
252
253 <YYINITIAL> {
254
255     /* Keywords */
256     "if" |
257     "else" |
258     "while" |
259     "for" |
260     "in" |
261     "func" |
262     "return" |
263     "null" { addToken(Token.RESERVED_WORD); }
264
265     /* Data types. */
266     "int" |
267     "real" |
268     "bool" |
269     "string" { addToken(Token.DATA_TYPE); }
270
271     /* Functions */
272     "print" |
273     "println" |

```

```
274     "read"                      { addToken(Token.FUNCTION); }
```

```
275  
276     {NewlineCharacter}          { addNullToken(); return firstToken; }
```

```
277  
278     {BooleanLiteral}           { addToken(Token.LITERAL_BOOLEAN); }
```

```
279  
280     {Identifier}              { addToken(Token.IDENTIFIER); }
```

```
281  
282     {Whitespace}              { addToken(Token.WHITESPACE); }
```

```
283  
284     /* String/Character Literals. */  
285     {RegularStringLiteral}      { addToken(Token.LITERAL_STRING_DOUBLE_QUOTE); }
```

```
286     {UnclosedRegularStringLiteral} { addToken(Token.ERROR_STRING_DOUBLE); addNullToken();  
return firstToken; }
```

```
287     {ErrorRegularStringLiteral} { addToken(Token.ERROR_STRING_DOUBLE); }
```

```
288  
289     /* Comments. */  
290     {SingleLineComment}         { addToken(Token.COMMENT_EOL); addNullToken(); return  
firstToken; }
```

```
291  
292     /* Separators. */  
293     {Separator}                { addToken(Token.SEPARATOR); }
```

```
294     {Separator2}               { addToken(Token.IDENTIFIER); }
```

```
295  
296     /* Operators. */  
297     {OperatorOrPunctuator}     { addToken(Token.OPERATOR); }
```

```
298  
299     /* Numbers */  
300     {DecimalIntegerLiteral}    { addToken(Token.LITERAL_NUMBER_DECIMAL_INT); }
```

```
301     {RealLiteral}              { addToken(Token.LITERAL_NUMBER_FLOAT); }
```

```
302     {ErrorNumberFormat}        { addToken(Token.ERROR_NUMBER_FORMAT); }
```

```
303  
304  
305     /* Pretty-much anything else. */  
306     {ErrorIdentifier}          { addToken(Token.ERROR_IDENTIFIER); }
```

```
307
308     /* Ended with a line not in a string or comment. */
309     <<EOF>>                      { addNullToken(); return firstToken; }
310
311     /* Catch any other (unhandled) characters and flag them as bad. */
312     .                           { addToken(Token.ERROR_IDENTIFIER); }
313
314 }
315
```