



DEPARTAMENTO DE SISTEMAS COMPUTACIONALES E INFORMÁTICA

ASUNTO: **Solicitud de Actividades**

Celaya, Gto., 2018-03-23

LENGUAJES Y AUTÓMATAS II

DOCENTE DESIGNADO: ISC. RICARDO GONZÁLEZ GONZÁLEZ

ACTIVIDAD 3 (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 PRIMERA ETAPA DEL ANÁLIZADOR LÉXICO 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 SEGUNDA FASE O ETAPA DEL PROYECTO, ES DECIR UN ANALIZADOR SINTÁCTICO.

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 LÉXICO 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.

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

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.



60
Aniversario
TecNM
en Celaya
1958-2018

Antonio García Cubas Pte. #600 esq. Av. Tecnológico Col. Alfredo V. Bonfil C.P. 38010
Celaya, Gto. AP 57, Comutador:(461)6117575, Correo electrónico: lince@itcelaya.edu.mx
www.itcelaya.edu.mx





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 SINTÁCTICO. ES DECIR, CREACIÓN DE ESTRUCTURAS DE DATOS COMO ÁRBOLES DE DERIVACIÓN SINTÁCTICA Y ALGORITMOS PARA DETERMINAR EL CUMPLIMIENTO EN EL ORDEN DE LOS ELEMENTOS QUE CONLLEVA ESTE ANÁLISIS, ETC.

LA TABLA DE SÍMBOLOS, SU CORRECTO DISEÑO Y SOBRE TODO SU APROPIADO LLENADO, SERÁ LA BASE DE ESTE SIGUIENTE EJERCICIO.

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

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 3 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. UNA TABLA DE SÍMBOLOS PERFECTAMENTE ESTRUCTURADA CON LA TOKENIZACIÓN APROPIADA Y SOLVENTADAS TODAS LAS OBSERVACIONES REALIZADAS POR EL PROFESOR DURANTE LA EVALUACIÓN PRESENCIAL.

C. PLANTEAMIENTO Y FUNDAMENTACIÓN DE LOS CASOS DE USO DEL ANALIZADOR SINTÁCTICO, ASI COMO LOS ERRORES EN QUE DERIVARÁ CADA UNO DE ELLOS.

D. 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.

E. GENERACIÓN DE UN CATÁLOGO DE ERRORES, CORRELACIONADO A LOS CASOS DE USO PROPUESTOS.

F. DISEÑO DE UNA SOLUCIÓN MODELADA EN OBJETOS, APOYADA EN DIAGRAMAS DE CLASE QUE DESCRIBAN LA ARQUITECTURA PROPUESTA PARA EL PROTOTIPO DEL ANALIZADOR SINTÁCTICO.

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



**H. MODELADO E IMPLEMENTACIÓN DE UN PROCESO :**

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 DE LOS CASOS DE ESTUDIOS IDENTIFICADOS Y PROPUESTOS.

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

J. GENERACIÓN DE UNA BITÁCORA DE INCIDENCIAS.

K. 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.

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



SEP

SECRETARÍA DE
EDUCACIÓN PÚBLICA



TECNOLÓGICO NACIONAL DE MÉXICO
en Celaya

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)**.**

A handwritten signature in blue ink.

A handwritten signature in blue ink.

A handwritten signature in blue ink.



60
Aniversario
TecNM
en Celaya
1956-2016

Antonio García Cubas Pte. #600 esq. Av. Tecnológico Col. Alfredo V. Bonfil C.P. 38010
Celaya, Gto. AP 57, Comutador:(461)6117575, Correo electrónico: lince@itcelaya.edu.mx
www.itcelaya.edu.mx



**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 APlican TODAS LAS MISMAS OBSERVACIONES ANTERIORES.



SEP

SECRETARÍA DE
EDUCACIÓN PÚBLICA



TECNOLÓGICO NACIONAL DE MÉXICO
en Celaya

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 CONSECUITIVO 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 :

2018-04-23_LAII_A3_EQUIPO_99_9999999_PEREZ_PEREZ_JUAN_ENE-JUN18.PDF

FECHA DE ENTREGA:

**VÍA CORREO ELECTRÓNICO, EL LUNES 23 DE ABRIL 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 3

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

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



60
Aniversario
TecNM
en Celaya
1958-2018

Antonio García Cubas Pte. #600 esq. Av. Tecnológico Col. Alfredo V. Bonfil C.P. 38010
Celaya, Gto. AP 57, Comutador:(461)6117575, Correo electrónico: lince@itcelaya.edu.mx

www.itcelaya.edu.mx



2A - El análisis sintáctico

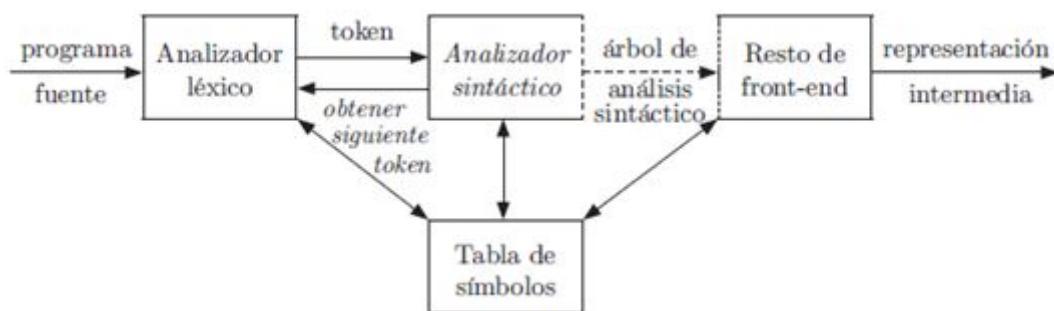
La Sintaxis es la rama de la gramática que estudia las relaciones de las palabras al combinarse para formar unidades superiores en significado.

¿Qué es el analizador Sintáctico?

El analizador sintáctico o parser, es la segunda fase del compilador. Esta fase utiliza los tokens adquiridos del analizador léxico junto con sus componentes respectivos, esto para poder generar una representación intermedia por medio de un árbol que ayudará a describir la estructura del flujo de tokens. Usualmente, es representado con un árbol sintáctico, donde cada nodo interno buscará la representación de una operación y sus nodos hijos serán los argumentos.

2B - ¿En qué consiste?

El analizador sintáctico obtiene una cadena de tokens del analizador léxico y verifica que la cadena de tokens pueda generarse mediante la gramática del lenguaje fuente. El analizador, se encarga de encontrar errores sintácticos en el código fuente y deberá identificarlos dentro del mismo. Como ya fue mencionado con anterioridad, el análisis utiliza un árbol sintáctico para representar la estructura de cada instrucción ingresada dentro de un análisis sintáctico, donde este árbol sintáctico se pasará al resto del compilador para que este realice los siguientes pasos de la compilación.



El analizador sintáctico puede ser visto en 3 tipos generales: Universal, Ascendente y Descendente. Los métodos universales, son llamados así ya que estos pueden analizar cualquier gramática, los métodos más conocidos son: "El algoritmo de Cocke-Younger-Kasami" y "El algoritmo de Earley". A pesar de que estos algoritmos son genéricos, estos son demasiado ineficientes para usarse en los compiladores.

Por lo regular, los métodos utilizados en los compiladores pueden clasificarse como Ascendentes o Descendentes. Según sus nombres, los métodos construyen el árbol sintáctico de maneras distintas. El Descendente construye el árbol de la parte superior (Raíz) a la parte inferior (hojas). El Ascendente parte de las hojas al nodo Raíz. Sin importar el método, la entrada al analizador se explora de izquierda a derecha, un símbolo a la vez.

2C - Casos de estudio

La falta del uso del delimitador ";" en las sentencias.

Supongamos que en una sentencia, no se usó el delimitador ";" al final de dicha sentencia.

```
print "Hola Mundo"
```

El analizador deberá marcar error cuando se encuentren estos casos. El número del error será el 210.

Return solo acepta un parámetro seguido de ;'

Este error se presenta cuando después del return se presenta algún token después de un primer parámetro que no sea punto y coma. Como por ejemplo:

```
return num * fact(num - 1);
```

En este caso después de haber puesto num, se debió de haber incluido el punto y coma. Por nuestra definición no reconoce la operación compleja.

Apertura y cierre de llaves o corchetes erróneas.

Si en el código fuente se encuentra una mala implementación de llaves y corchetes, con base a nuestro lenguaje estos serán reconocidos como errores, el error correspondiente será el 230.

Por ejemplo:

```
Metodo(suma); // La llamada a un método sin el cierre de paréntesis.
```

Mala estructuración de sentencias con base a nuestro lenguaje definido

Dentro de nuestro lenguaje se definió la estructura que se debe llevar para los distintos elementos que formarán parte del código fuente, tales como las sentencias, las expresiones, los métodos, entre otros.

Sin embargo, si alguna de estas ya mencionadas no tuviera la estructura correspondiente, este deberá ser reconocido como un error sintáctico, el error correspondiente será el 240.

Ejemplo:

```
if(avión 98){}
```

Este fragmento de código esta mal, ya que en nuestro lenguaje se pide que dentro de la estructura del if, debe constar de una expresión relacional, la cual no está presente.

Para que fuera válida, debería estar definido de la siguiente manera.

```
if(avión == 98){}
```

Estructuras no definidas en el lenguaje

Este es un caso en el que directamente hay una sucesión de tokens que no se asemejan a ninguna estructura definida en el lenguaje. Este sería el caso que incluiría toda sentencia no válida no incluida en los casos anteriores.

Por ejemplo la siguiente sucesión:

```
gato juan alberto
```

```
3 4 + 4 - gato == perro
```

Aunque sean todos tokens reconocidos, no corresponden a ninguna estructura; Por lo que si se presenta este caso, deberá ser reconocido como un error el cual se identificara como el error 250.

```
#####
#           CASO DE ESTUDIO 1 y 2
#       La falta del uso del delimitador ";" en las sentencias
#       Return solo acepta un parámetro seguido de punto y coma
#####

func fact (int num): int {
    if (num == 1) {
        return 1 //la falta de ";" hace que esta sentencia
    } else { //sea incorrecta
        return num * fact(num - 1); //Dentro de un Return, no se aceptan
                                    //realizar operaciones.
    }
}

func fib (int num): int {
    if (num == 1) {
        return 1;
    } else if (num == 2) {
        return 1 //de nuevo, hace falta el ;;
    } else {
        return fib(num - 1) + fib (num - 2); //Este return tambien hace
                                                //operaciones.
    }
}

int num = 0 //de nuevo, hace falta el ;
int fib; //Aqui es un error ya que no hay definida
          //una estructura de declaración

int fact;
println "Dame un numero para hacer unos calculos:"
print ">" //de nuevo, hace falta el ;
read num;
fact = fact(num);
fib = fib (num) //de nuevo, hace falta el ;
println "El factorial de " + num + " es " + fact;
println "El numero de fibonacci de " + num + " es " + fib;
//los print no estan bien estructurados segun la definición
```

```
#####
#           CASO DE ESTUDIO 3, 4 y 5          #
#           Mal cierre de paréntesis          #
#           Mala estructuración de sentencias   #
#           Estructuras no definidas          #
#####

func sum (int num1 , int num2) : int { // La estructura del método es la
                                         //definida con base en nuestra
                                         //gramática.
    int suma;
    suma = num1 + num2;
    return suma;                         //return esta estructurado correctamente
}

func res (int num1  int num2 int) : int { //Esta estructura no cumple con la
                                         //gramática por lo que marcaría error
                                         //ya que hace falta las comas y falta
                                         //una variable.
    int resta;
    resta = num1 - num2;
    return resta num1;                  //Este return no esta estructurado correctamente
                                         //por lo que enviaría un error.
}

if(suma>resta){                      //La estructura de este if
                                         //es la correcta.
    print ("La suma es mayor que la resta"); //Ningún print lleva paréntesis
}else{
    print ("La resta es mayor que la suma");
}

if(suma>resta {                      //La estructura de este if
                                         //es incorrecta por lo del
                                         //paréntesis y el else x
    print ("La suma es mayor que la resta");
    print ("Esto es error");
}else x{
    print ("La resta es mayor que la suma");
    print ("Esto es un error");
}
el perro es un animal;                //Es una estructura no definida
                                         //el lenguaje

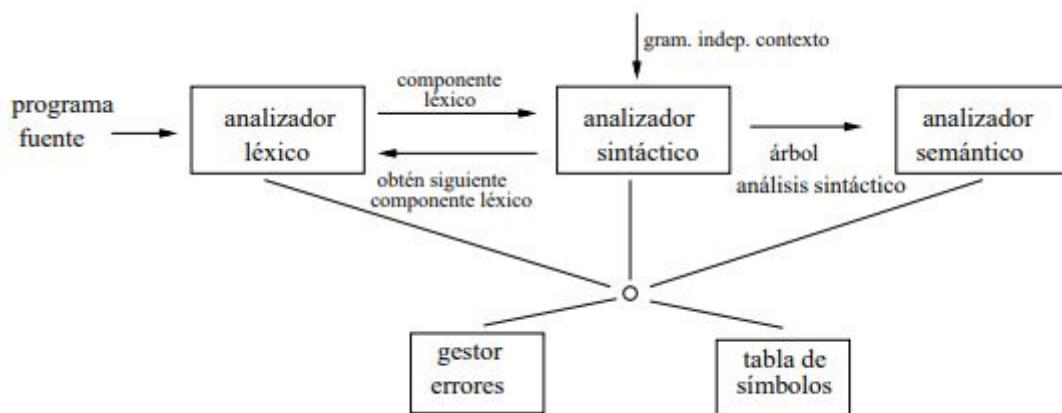
int num1 = 15;
int num2 02 x 15u;      //Enviara un error semantico
real num3 = 15.02;
real num4 >= 16,42;
int suma = sum (num1 , num2);
int resta = res (num1 , num2);
real multi = mult (num3 num4);    //Enviara un error semántico
real divi = div (num3 , num4);
print (suma , resta , multi , divi);
```

2D - Procesos

Hay tres procesos o funciones principales de un analizador sintáctico. Estos son:

- La primera funcionalidad y la principal, es reconocer si la cadena de tokens enviada por el analizador léxico y que es tomada de la tabla de símbolos, puede ser formada por las reglas de producción definidas en la gramática del lenguaje.
- Su segunda funcionalidad es la de generar el árbol de derivación sintáctica. Este árbol es el que sirve para generar la cadena de tokens y se usa como representación intermedia en la generación del código.
- Informar de los errores generados durante el análisis. Los errores pueden ser manejados con una pila de errores.

Diagrama de los procesos sintácticos:



Para realizar estas funciones se propone el uso de una estructura de árbol, donde cada nodo tenga los siguientes campos: Nodo Padre, Arreglo de Nodos Hijos, Referencia a la tabla de símbolos, Id del Token y Id de la Regla de Producción que usa. Otra estructura que se usará será la pila de errores.

Se propone usar métodos recursivos para la verificación de las cadenas de tokens según la gramática definida.

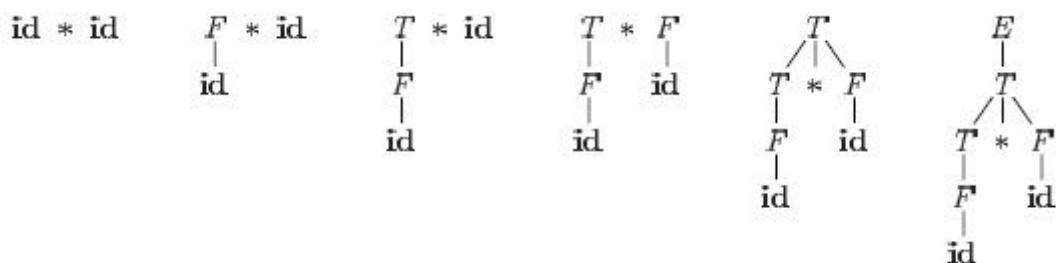
También se propone hacer la validación y la generación del árbol de manera simultánea.

2E - ¿Cómo se puede realizar la implementación?

Como ya se abordó en los puntos anteriores, existen 3 tipos de analizadores sintácticos que pueden implementarse: Universales, Ascendentes y Descendentes. La implementación estará ligada a la definición del tipo de analizador que se llevará a cabo. Se menciona en nuestra fuente principal de información que en el tipo universal de los analizadores no es muy utilizado, ya que llega a ser ineficiente, siendo los analizadores más comunes los ascendentes y descendentes.

Ascendentes:

Al análisis sintáctico ascendente como lo dice su nombre, es la construcción de un árbol sintáctico a partir del código fuente donde se comienza en las hojas y este avanza hasta la raíz del arbol.



El análisis ascendente puede ser considerado como un proceso para reducir una cadena al símbolo inicial de la gramática. Durante la reducción se sustituye un fragmento del código fuente que coincide con el cuerpo de una producción, por el no terminal que se encuentra en la producción.

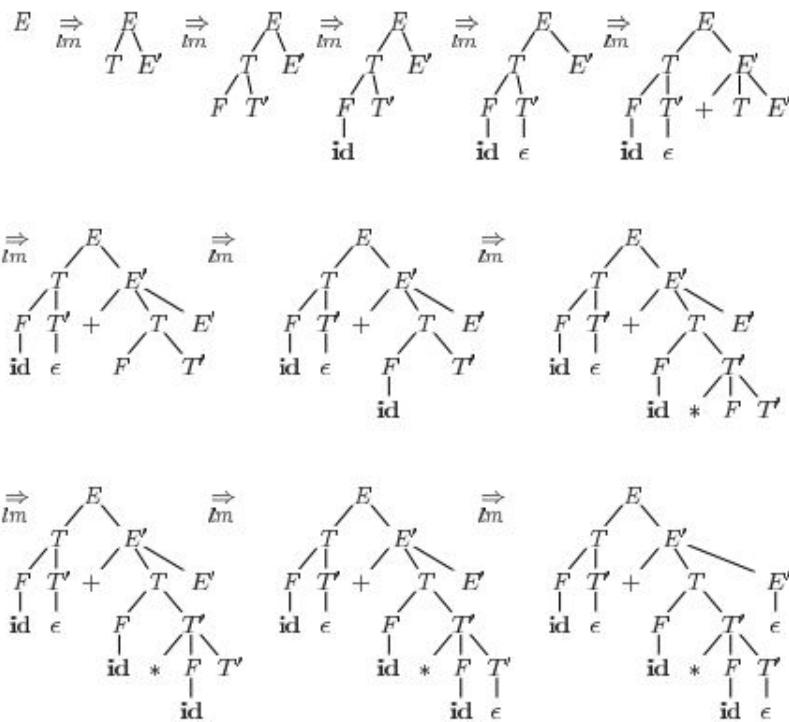
$\text{id} * \text{id}, F * \text{id}, T * \text{id}, T * F, T, E$

Descendentes:

El análisis descendente se basa en la construcción de un árbol de análisis sintáctico. Este parte desde la raíz y crea los nodos del árbol de análisis sintáctico en preorden (profundidad primero).

Esto se puede identificar en las siguientes dos imágenes, en la primera de ellas se propone una gramática y la segunda de ellas es el árbol de análisis sintáctico para formar la entrada **id + id * id**.

$$\begin{array}{lcl}
 E & \rightarrow & TE' \\
 E' & \rightarrow & + TE' \mid \epsilon \\
 T & \rightarrow & FT' \\
 T' & \rightarrow & *FT' \mid \epsilon \\
 F & \rightarrow & (E) \mid \text{id}
 \end{array}$$



Este consiste en un conjunto de procedimientos para cada no terminal, se va resolviendo hasta llegar a un elemento terminal y se hace un desplazamiento hacia atrás para seguir resolviendo los no terminales, hasta que ya no haya elementos no terminales que expandir.

Referencias bibliográficas

AHO, ALFRED V. (2008). COMPILEDORES. PRINCIPIOS, TÉCNICAS Y HERRAMIENTAS. MÉXICO: PEARSON EDUCACIÓN.

<http://informatica.uv.es/docencia/iiguia/asignatu/2000/PL/2007/tema3.pdf>

<http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/Cuaderno-61-Matematicas-Sintactico.pdf>

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”

Se definirán en la última etapa.

Estructuras

En la siguiente tabla se presentan las estructuras que corresponden a las reglas de producción que definen nuestro lenguaje, junto con su ID y la secuencia inicial que servirá para identificar cada estructura, si es que aplica.

ESTRUCTURA	SECUENCIA INICIAL (si aplica)	ID
R_BLOQUE		100
R_CALL	T_FUN, T_LPAR ...	101
R_METODO	T_FUNC ...	102
R_PARAMS		103
R_ARGS		104
R_RETORNO	T_RETURN ...	105
R_IF	T_IF ...	106
R_FOR	T_FOR ...	107
R_WHILE	T WHILE ...	108
R_LECTURA	T_READ ...	109
R_IMPRESION	T_PRINT ...	110
R_ASIGNACION	[(T_INT T_REAL T_STRING T_BOOL),] T_VAR, T_ASSIGN ...	111
R_EXPR_REL		112
R_EXPR_ARITM		113

Tabla de símbolos

Con respecto a la fase anterior, se realizaron algunos cambios a la tabla de símbolos, quedando sus respectivos campos como se describe a continuación:

Registro de la tabla de símbolos

CAMPO	DESCRIPCIÓN
String ID	ID único para la identificación de un lexema.
String NOMBRE	Es el nombre o lexema que será almacenado en la tabla de símbolos.
int TOKEN_ID	Es el identificador del token que ha sido reconocido, de acuerdo al catálogo.
int TIPO_DATO	El tipo de dato del lexema si es identificador. Si el identificador ya existe, el tipo se marca como referencia. Si es una función, lo que regresa. Se usará en las siguientes etapas.
String VALOR	Este campo puede tener el ID de una referencia, o el valor inicial de una variable.
int LINE	Línea del código fuente en la que se encuentra el token.

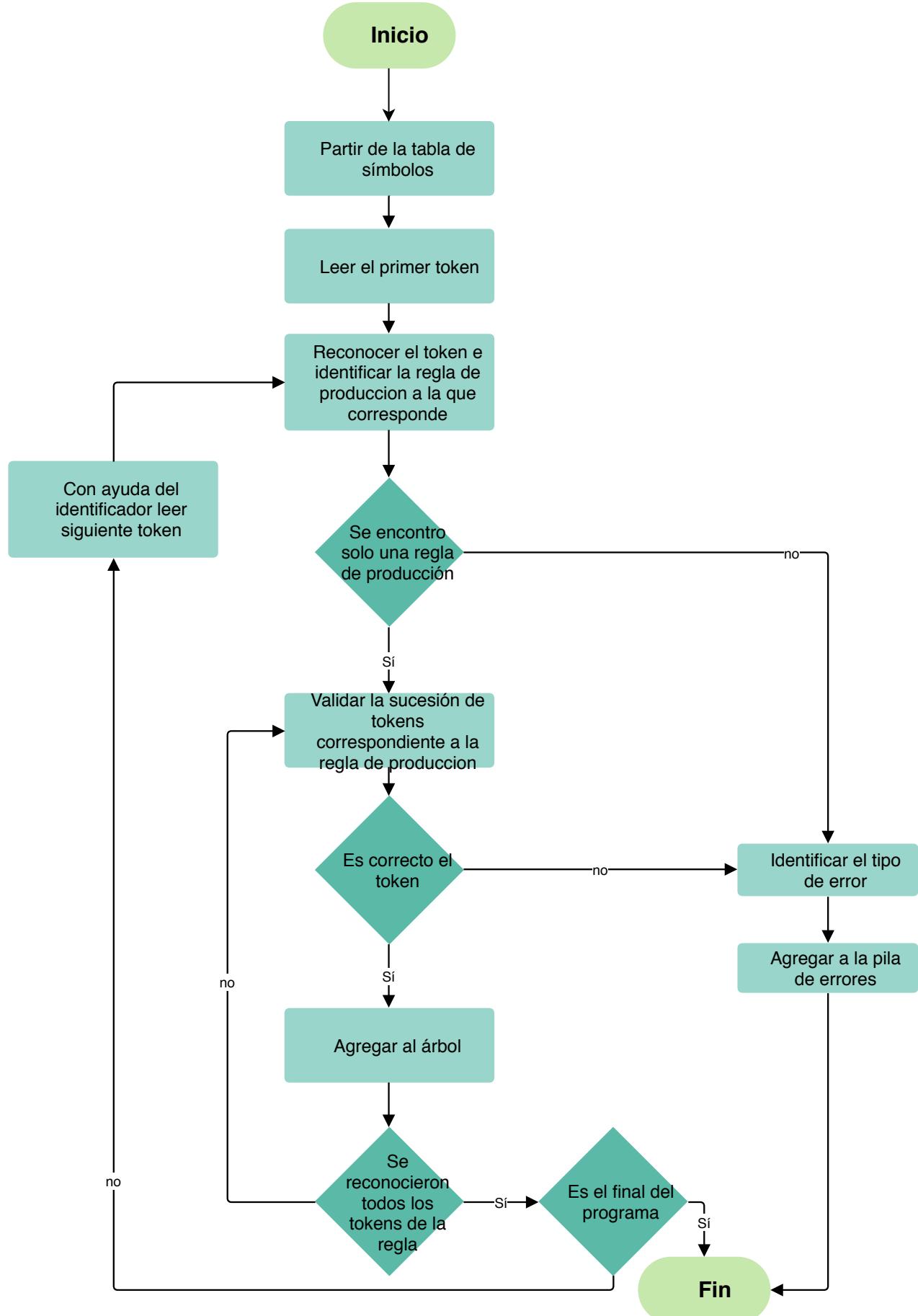
Árbol de derivación sintáctico.

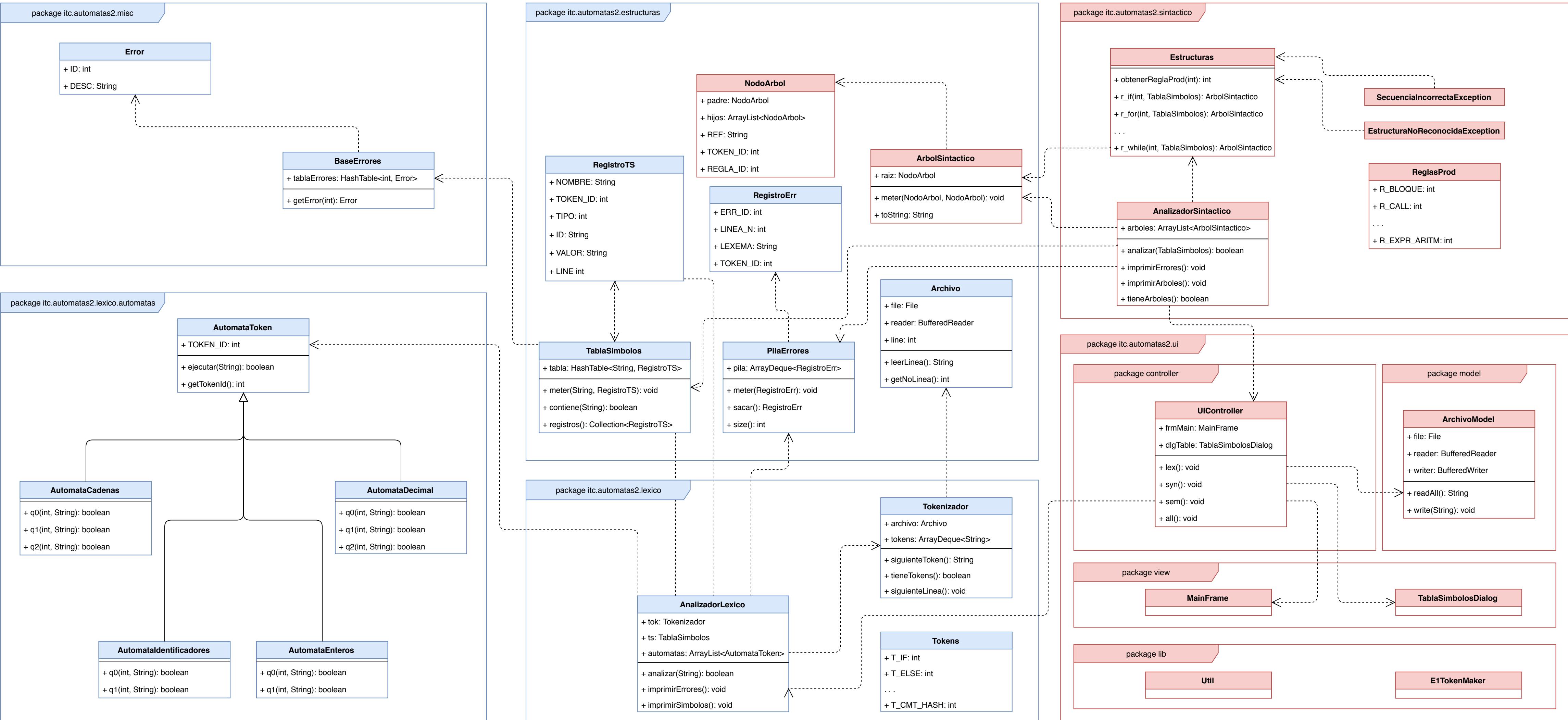
En el árbol de derivación se contendrán los tokens de una sentencia según las estructuras definidas en el lenguaje, para su posterior análisis en la siguiente etapa. El analizador sintáctico deberá de construirlo si la secuencia de tokens es correcta.

El árbol se compone de nodos enlazados entre ellos. Los nodos contendrán la información que se describe a continuación:

Nodo del Árbol de derivación sintáctico

CAMPO	DESCRIPCIÓN
Nodo PADRE	Es el nodo superior con el cual tiene una relación.
ArrayList<Nodo> HIJOS	Son los nodos inferiores con los cuales tiene una relación.
String REF	Si el nodo corresponde a un terminal (token), es el ID del mismo en la tabla de símbolos. De esta forma, se puede obtener información del token correspondiente.
int TOKEN_ID	Si el nodo corresponde a un terminal, es el identificador del token que se encuentra en la tabla de símbolos.
int REGLA_ID	Si el nodo no corresponde a un terminal, es el identificador de la regla de producción a la que pertenece.





EQUIPO 1		ACTIVIDAD 3															LENGUAJES Y AUTÓMATAS II				
ACTIVIDADES / FASES		TIEMPOS POR ACTIVIDAD															TOTAL POR ACTIVIDAD		OBSERVACIONES		
		07/04	08/04	09/04	10/04	11/04	12/04	13/04	14/04	15/04	16/04	17/04	18/04	19/04	20/04	21/04	22/04	23/04	PLANEADO	REAL	
Correcciones de la actividad 2																					
Correcciones al diseño		240																240	360	Durante la implementación surgieron errores no contemplados. Se corrigieron.	
Correcciones al código			240	120	120													240	720	Ver bitácora de incidencias.	
			120	120	120	120											120	120			
Actividades iniciales																					
Lectura inicial de la solicitud de actividad				60													60	60			
Investigación del marco teórico y discusión acerca de los temas					240												240	360	Surgieron dudas y conflictos acerca del marco teórico. Ver bitácora de incidencias.		
					120	120	120										120	120			
Análisis																					
Planteamiento del problema					120												120	120			
Documentación del marco teórico						120											120	240			
																	120	240			
Diseño																					
Definición de casos de estudio						120											120	120	A partir del 18/04 se trabajó a marchas forzadas debido al retraso.		
Categorización de errores						120											120	120			
Diagramas de flujo							480										480	600			
Diagramas de clases								480									480	360			
																	480	360			
Construcción																					
Implementación de componentes								480	480	480	480	240					2,160	2,520			
Integración de componentes									120	240	240	960	960				960	240	A diferencia de la etapa pasada, la integración se efectuó de manera rápida.		
																	960	240			
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		
																	15	15	15	15	
			</td																		

1. Por decisión del equipo en conjunto, las semanas santa y de pascua se tomaron como periodo vacacional, comenzando trabajos el sábado 7 de abril.

07 de abril del 2018

2. Mientras se hacían las correcciones sugeridas, se detectaron casos muy especiales que producían fallas en el análisis léxico, las cuales se solucionaron a tiempo. De igual manera, la tokenización de una cadena ya no depende de los delimitadores para identificar símbolos especiales (símbolos considerados como un solo token).

10 de abril del 2018

3. Durante la fase de investigación y conceptualización surgieron dudas significativas que nos impidieron continuar con el trabajo. Se decidió avanzar parcialmente en la etapa del diseño (diagramas básicos, estructuras de datos, interfaz gráfica, entre otros).
4. Se tuvo reunión con el profesor y se resolvieron todas las dudas que surgieron.

18 de abril del 2018

5. Durante la implementación, se detectaron errores en la caracterización del lenguaje y el catálogo de tokens. Se hicieron las modificaciones necesarias.

19-22 de abril del 2018

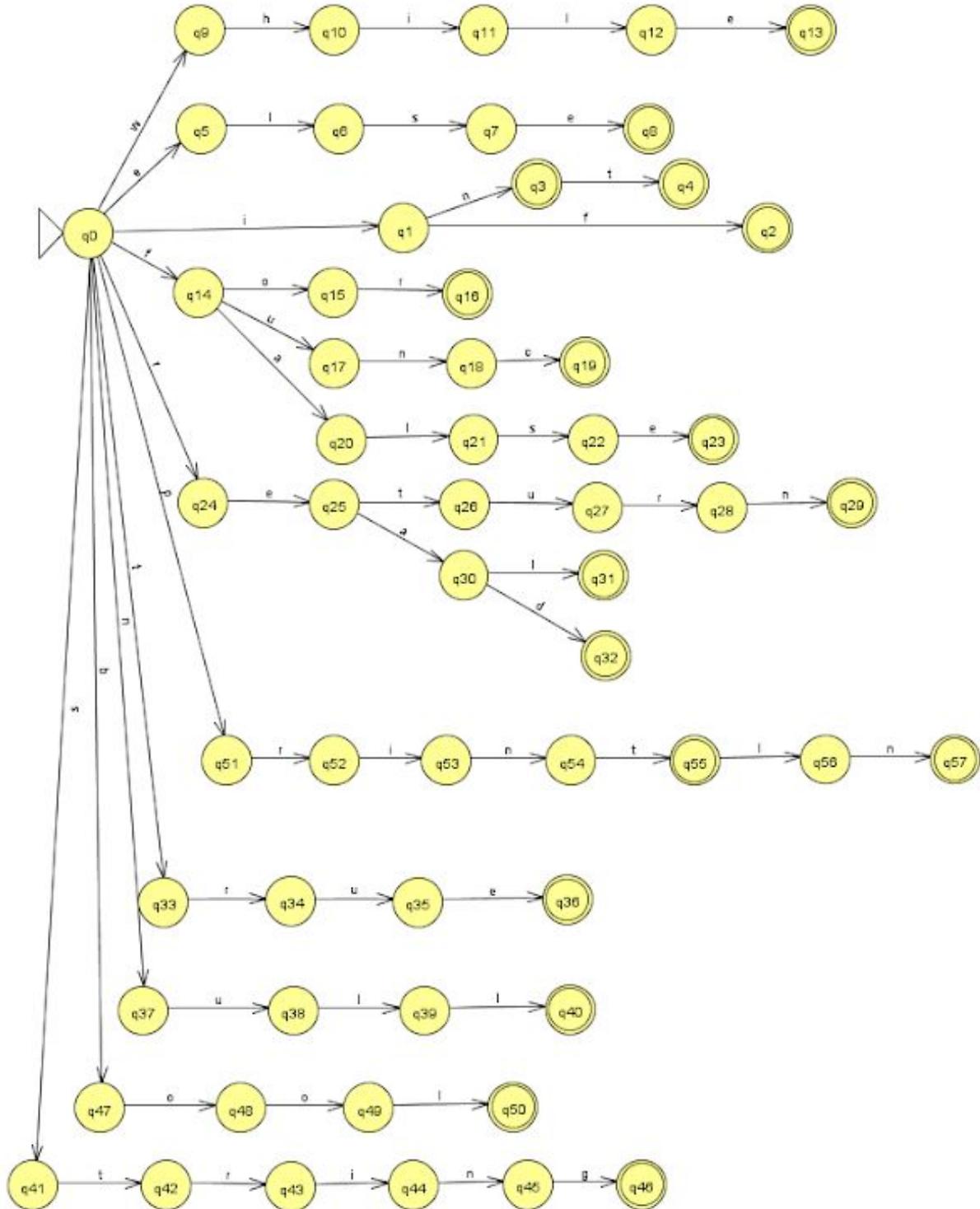
6. En general, durante la investigación acerca del marco teórico, el equipo se sintió inseguro de continuar. Debido a esto, la mayoría de las actividades planeadas se retrasaron varios días y se tuvo que trabajar a marchas forzadas, cosa que se ve reflejada en la calendarización.

23 de abril del 2018

Anexo

Correcciones realizadas
al producto anterior

Para tener un análisis léxico correcto, se desarrolló el siguiente autómata que identifica las palabras reservadas del lenguaje.



Con respecto al catálogo de tokens, se hicieron algunas modificaciones, las cuales son resaltadas a continuación.

Palabras reservadas

IDs del 1 al 17

TOKEN	LEXEMA/REGEX	ID
T_IF	if	1
T_ELSE	else	2
T WHILE	while	3
T FOR	for	4
T_IN	in	5
T_FUNC	func	6
T_RETURN	return	7
T_TRUE	true	8
T_FALSE	false	9
T_NULL	null	10
T_INT	int	11
T_REAL	real	12
T_BOOL	bool	13
T_STRING	string	14
T_PRINT	print	15
T_PRINTF	printf	16
T_READ	read	17

Identificadores y constantes

IDs del 18 al 23

TOKEN	LEXEMA/REGEX	ID
T_VAR	[_ \$a-Z][_ \$a-Z0-9]*	18
T_FUN	[_ \$a-Z][_ \$a-Z0-9]*\(\)	19
T_INT_CONST	[0-9][0-9]*	20
T_REAL_CONST	[0-9][0-9]*\.[0-9][0-9]*	21
T_BOOL_CONST	true false	22
T_STR_CONST	\“[\x20-\x7E]*\”	23

Símbolos y operadores

IDs del 24 al 47

TOKEN	LEXEMA/REGEX	ID
T_COLON	:	24
T_SEMICOLON	;	25
T_LPAREN	(26
T_RPAREN)	27
T_LBRACE	{	28
T_RBRACE	}	29
T_LBRACKET	[30
T_RBRACKET]	31
T_LTE	<=	32
T_GTE	>=	33

T_NE	!=	34
T_LT	<	35
T_GT	>	36
T_EQ	==	37
T_ASSIGN	=	38
T_DEC	--	39
T_INC	++	40
T_NOT	!	41
T_PLUS	+	42
T_MINUS	-	43
T_STAR	*	44
T_DIV	/	45
T_RANGE	->	46
T_COMMA	,	47

```
<bloque> ::= "{" <sentencias> "}"  
  
<sentencias> ::= <sentencia> <sentencia>*  
<sentencia> ::= <asignación> | <llamada método> | <ciclos> |  
<condicionales> | <lectura> | <impresión> | <return>  
  
<llamada a método> ::= <id> "(" "[<parámetros>]"")"  
<método> ::= func <id> "(" "[<argumentos>]"")" [: <tipo>] "{" <sentencias> "}"  
<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>;  
<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>
```

Anexo

Código generado
durante esta etapa

```
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 secuencia.
18     * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
19     * estructura dentro de un bloque.
20     */
21    public static ArbolSintactico r_if(int i, TablaSimbolos ts) throws SecuenciaIncorrectaException,
22    EstructuraNoReconocidaException {
23        cont = i;
24        ArbolSintactico a = new ArbolSintactico();
25        RegistroTS reg;
26        NodoArbol actual;
27        actual = new NodoArbol();
28        actual.REGLA_ID = ReglasProd.R_IF;
29        a.raiz = actual;
30
31        reg = ts.get(cont);
32        actual = new NodoArbol();
33        actual.TOKEN_ID = Tokens.T_IF;
34        actual.REF = reg.ID;
35        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     Arbolesintactico 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
84 estructura dentro de un bloque.
85 */
86 private static ArbolSintactico r_else(TablaSimbolos ts) throws SecuenciaIncorrectaException,
87 EstructuraNoReconocidaException {
88     RegistroTS reg = ts.get(cont);
89     if (reg.TOKEN_ID == Tokens.T_IF) {
90         return r_if(cont, ts);
91     } else {
92         return r_bloque(cont, ts);
93     }
94 }
95 /**
96 * Analiza la estructura sintáctica del bloque y regresa su árbol
97 *
98 * @param i el índice del primer token en la tabla
99 * @param ts la tabla de símbolos
100 * @return el árbol sintáctico de la estructura
101 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
102 * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
```

```
101 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);
```

```
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     }
166     reg = ts.get(cont);
167 } while (reg.TOKEN_ID != Tokens.T_RBRACE);
168
169 actual = new NodoArbol();
```

```
170         actual.TOKEN_ID = Tokens.T_RBRACE;
171         actual.REF = reg.ID;
172         a.meter(a.raiz, actual);
173
174         cont++;
175         return a;
176     }
177
178     /**
179      * Analiza la estructura sintáctica del call y regresa su árbol
180      *
181      * @param i el índice del primer token en la tabla
182      * @param ts la tabla de símbolos
183      * @return el árbol sintáctico de la estructura
184      * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
185      */
186     public static ArbolSintactico r_call(int i, TablaSimbolos ts) throws
187     SecuenciaIncorrectaException {
188         cont = i;
189         ArbolSintactico a = new ArbolSintactico();
190         RegistroTS reg;
191         NodoArbol actual;
192         actual = new NodoArbol();
193         actual.REGLA_ID = ReglasProd.R_CALL;
194         a.raiz = actual;
195
196         reg = ts.get(cont);
197         actual = new NodoArbol();
198         actual.TOKEN_ID = Tokens.T_FUN;
199         actual.REF = reg.ID;
200         a.meter(a.raiz, actual);
201
202         reg = ts.get(++cont);
203         if (reg.TOKEN_ID != Tokens.T_LPAREN) {
204             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
```

```
204         throw new SecuenciaIncorrectaException();
205     }
206     actual = new NodoArbol();
207     actual.TOKEN_ID = Tokens.T_LPAREN;
208     actual.REF = reg.ID;
209     a.meter(a.raiz, actual);
210
211     reg = ts.get(++cont);
212     if (reg.TOKEN_ID != Tokens.T_RPAREN) {
213         ArbolSintactico a2 = r_params(cont, ts);
214         a.meter(a.raiz, a2.raiz);
215     }
216     reg = ts.get(cont);
217     if (reg.TOKEN_ID != Tokens.T_RPAREN) {
218         PilaErrores.meter(new RegistroErr(230, reg.LINE, reg.NOMBRE));
219         throw new SecuenciaIncorrectaException();
220     }
221     actual = new NodoArbol();
222     actual.TOKEN_ID = Tokens.T_RPAREN;
223     actual.REF = reg.ID;
224     a.meter(a.raiz, actual);
225     cont++;
226     return a;
227 }
228
229 /**
230 * Analiza la estructura sintáctica del metodo y regresa su árbol
231 *
232 * @param i el índice del primer token en la tabla
233 * @param ts la tabla de símbolos
234 * @return el árbol sintáctico de la estructura
235 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
236 * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
237 * estructura dentro de un bloque.
238 */
```

```
238     public static ArbolSintactico r_metodo(int i, TablaSimbolos ts) throws
239         SecuenciaIncorrectaException, EstructuraNoReconocidaException {
240         cont = i;
241         ArbolSintactico a = new ArbolSintactico();
242         RegistroTS reg;
243         NodoArbol actual;
244         actual = new NodoArbol();
245         actual.REGLA_ID = ReglasProd.R_METODO;
246         a.raiz = actual;
247
248         reg = ts.get(cont);
249         actual = new NodoArbol();
250         actual.TOKEN_ID = Tokens.T_FUNC;
251         actual.REF = reg.ID;
252         a.meter(a.raiz, actual);
253
254         reg = ts.get(++cont);
255         if (reg.TOKEN_ID != Tokens.T_FUN) {
256             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
257             throw new SecuenciaIncorrectaException();
258         }
259         actual = new NodoArbol();
260         actual.TOKEN_ID = Tokens.T_FUN;
261         actual.REF = reg.ID;
262         a.meter(a.raiz, actual);
263
264         reg = ts.get(++cont);
265         if (reg.TOKEN_ID != Tokens.T_LPAREN) {
266             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
267             throw new SecuenciaIncorrectaException();
268         }
269         actual = new NodoArbol();
270         actual.TOKEN_ID = Tokens.T_LPAREN;
271         actual.REF = reg.ID;
272         a.meter(a.raiz, actual);
```

```
272  
273     reg = ts.get(++cont);  
274     if (reg.TOKEN_ID != Tokens.T_RPAREN) {  
275         Arbolesintactico a2 = r_args(cont, ts);  
276         a.meter(a.raiz, a2.raiz);  
277     }  
278     reg = ts.get(cont);  
279     if (reg.TOKEN_ID != Tokens.T_RPAREN) {  
280         PilaErrores.meter(new RegistroErr(230, reg.LINE, reg.NOMBRE));  
281         throw new SecuenciaIncorrectaException();  
282     }  
283     actual = new NodoArbol();  
284     actual.TOKEN_ID = Tokens.T_RPAREN;  
285     actual.REF = reg.ID;  
286     a.meter(a.raiz, actual);  
287  
288     reg = ts.get(++cont);  
289     if (reg.TOKEN_ID == Tokens.T_COLON) {  
290         actual = new NodoArbol();  
291         actual.TOKEN_ID = Tokens.T_COLON;  
292         actual.REF = reg.ID;  
293         a.meter(a.raiz, actual);  
294         reg = ts.get(++cont);  
295         switch (reg.TOKEN_ID) {  
296             case Tokens.T_INT:  
297             case Tokens.T_REAL:  
298             case Tokens.T_BOOL:  
299             case Tokens.T_STRING:  
300                 actual = new NodoArbol();  
301                 actual.TOKEN_ID = reg.TOKEN_ID;  
302                 actual.REF = reg.ID;  
303                 a.meter(a.raiz, actual);  
304                 break;  
305             default:  
306                 PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
307             }  
308         }  
309     }  
310     reg = ts.get(cont);  
311     if (reg.TOKEN_ID != Tokens.T_LPAREN) {  
312         PilaErrores.meter(new RegistroErr(250, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
313     }  
314     a.meter(a.raiz, actual);  
315 }
```

```
307                     throw new SecuenciaIncorrectaException();  
308             }  
309             cont++;  
310         }  
311         ArbolSintactico a2 = r_bloque(cont, ts);  
312         a.meter(a.raiz, a2.raiz);  
313         return a;  
314     }  
315  
316     /**  
317      * Analiza la estructura sintáctica del retorno y regresa su árbol  
318      *  
319      * @param i el índice del primer token en la tabla  
320      * @param ts la tabla de símbolos  
321      * @return el árbol sintáctico de la estructura  
322      * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.  
323      */  
324     public static ArbolSintactico r_retorno(int i, TablaSimbolos ts) throws  
SecuenciaIncorrectaException {  
325         cont = i;  
326         ArbolSintactico a = new ArbolSintactico();  
327         RegistroTS reg;  
328         NodoArbol actual;  
329         actual = new NodoArbol();  
330         actual.REGLA_ID = ReglasProd.R_RETORNO;  
331         a.raiz = actual;  
332  
333         reg = ts.get(cont);  
334         actual = new NodoArbol();  
335         actual.TOKEN_ID = Tokens.T_RETURN;  
336         actual.REF = reg.ID;  
337         a.meter(a.raiz, actual);  
338  
339         reg = ts.get(++cont);  
340         switch (reg.TOKEN_ID) {
```

```
341     case Tokens.T_VAR:
342     case Tokens.T_INT_CONST:
343     case Tokens.T_REAL_CONST:
344     case Tokens.T_BOOL_CONST:
345     case Tokens.T_STR_CONST:
346         actual = new NodoArbol();
347         actual.TOKEN_ID = reg.TOKEN_ID;
348         actual.REF = reg.ID;
349         a.meter(a.raiz, actual);
350         break;
351     default:
352         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
353         throw new SecuenciaIncorrectaException();
354     }
355
356     reg = ts.get(++cont);
357     if (reg.TOKEN_ID != Tokens.T_SEMICOLON) {
358         PilaErrores.meter(new RegistroErr(220, reg.LINE, reg.NOMBRE));
359         throw new SecuenciaIncorrectaException();
360     }
361     actual = new NodoArbol();
362     actual.TOKEN_ID = Tokens.T_SEMICOLON;
363     actual.REF = reg.ID;
364     a.meter(a.raiz, actual);
365     cont++;
366     return a;
367 }
368
369 /**
370 * Analiza la estructura sintáctica del for y regresa su árbol
371 *
372 * @param i el índice del primer token en la tabla
373 * @param ts la tabla de símbolos
374 * @return el árbol sintáctico de la estructura
375 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
```

```
376     * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o  
377     * estructura dentro de un bloque.  
378     */  
379     public static ArbolSintactico r_for(int i, TablaSimbolos ts) throws  
380     SecuenciaIncorrectaException, EstructuraNoReconocidaException {  
381         cont = i;  
382         ArbolSintactico a = new ArbolSintactico();  
383         RegistroTS reg;  
384         NodoArbol actual;  
385         actual = new NodoArbol();  
386         actual.REGLA_ID = ReglasProd.R_FOR;  
387         a.raiz = actual;  
388  
389         reg = ts.get(cont);  
390         actual = new NodoArbol();  
391         actual.TOKEN_ID = Tokens.T_FOR;  
392         actual.REF = reg.ID;  
393         a.meter(a.raiz, actual);  
394  
395         reg = ts.get(++cont);  
396         if (reg.TOKEN_ID != Tokens.T_LPAREN) {  
397             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
398             throw new SecuenciaIncorrectaException();  
399         }  
400         actual = new NodoArbol();  
401         actual.TOKEN_ID = Tokens.T_LPAREN;  
402         actual.REF = reg.ID;  
403         a.meter(a.raiz, actual);  
404  
405         reg = ts.get(++cont);  
406         if (reg.TOKEN_ID != Tokens.T_VAR) {  
407             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
408             throw new SecuenciaIncorrectaException();  
409         }  
410         actual = new NodoArbol();
```

```
409     actual.TOKEN_ID = Tokens.T_VAR;
410     actual.REF = reg.ID;
411     a.meter(a.raiz, actual);
412
413     reg = ts.get(++cont);
414     if (reg.TOKEN_ID != Tokens.T_IN) {
415         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
416         throw new SecuenciaIncorrectaException();
417     }
418     actual = new NodoArbol();
419     actual.TOKEN_ID = Tokens.T_IN;
420     actual.REF = reg.ID;
421     a.meter(a.raiz, actual);
422
423     reg = ts.get(++cont);
424     switch (reg.TOKEN_ID) {
425         case Tokens.T_VAR:
426         case Tokens.T_INT_CONST:
427             actual = new NodoArbol();
428             actual.TOKEN_ID = reg.TOKEN_ID;
429             actual.REF = reg.ID;
430             a.meter(a.raiz, actual);
431             break;
432         default:
433             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
434             throw new SecuenciaIncorrectaException();
435     }
436
437     reg = ts.get(++cont);
438     if (reg.TOKEN_ID != Tokens.T_RANGE) {
439         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
440         throw new SecuenciaIncorrectaException();
441     }
442     actual = new NodoArbol();
443     actual.TOKEN_ID = Tokens.T_RANGE;
```

```
444         actual.REF = reg.ID;
445         a.meter(a.raiz, actual);
446
447         reg = ts.get(++cont);
448         switch (reg.TOKEN_ID) {
449             case Tokens.T_VAR:
450             case Tokens.T_INT_CONST:
451                 actual = new NodoArbol();
452                 actual.TOKEN_ID = reg.TOKEN_ID;
453                 actual.REF = reg.ID;
454                 a.meter(a.raiz, actual);
455                 break;
456             default:
457                 PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
458                 throw new SecuenciaIncorrectaException();
459         }
460
461         reg = ts.get(++cont);
462         if (reg.TOKEN_ID != Tokens.T_RPAREN) {
463             PilaErrores.meter(new RegistroErr(230, reg.LINE, reg.NOMBRE));
464             throw new SecuenciaIncorrectaException();
465         }
466         actual = new NodoArbol();
467         actual.TOKEN_ID = Tokens.T_RPAREN;
468         actual.REF = reg.ID;
469         a.meter(a.raiz, actual);
470
471
472         ArbolSintactico a2 = r_bloque(++cont, ts);
473         a.meter(a.raiz, a2.raiz);
474         return a;
475     }
476
477 /**
478 * Analiza la estructura sintáctica del while y regresa su árbol
```

```
479     *
480     * @param i el índice del primer token en la tabla
481     * @param ts la tabla de símbolos
482     * @return el árbol sintáctico de la estructura
483     * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
484     * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
485     * estructura dentro de un bloque.
486     */
487     public static ArbolSintactico r_while(int i, TablaSimbolos ts) throws
488     SecuenciaIncorrectaException, EstructuraNoReconocidaException {
489
490         cont = i;
491         ArbolSintactico a = new ArbolSintactico();
492         RegistroTS reg;
493         NodoArbol actual;
494         actual = new NodoArbol();
495         actual.REGLA_ID = ReglasProd.R_WHILE;
496         a.raiz = actual;
497
498         reg = ts.get(cont);
499         actual = new NodoArbol();
500         actual.TOKEN_ID = Tokens.T WHILE;
501         actual.REF = reg.ID;
502         a.meter(a.raiz, actual);
503
504         reg = ts.get(++cont);
505         if (reg.TOKEN_ID != Tokens.T_LPAREN) {
506             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
507             throw new SecuenciaIncorrectaException();
508         }
509         actual = new NodoArbol();
510         actual.TOKEN_ID = Tokens.T_LPAREN;
511         actual.REF = reg.ID;
512         a.meter(a.raiz, actual);
```

```
512     ArbolSintactico a2 = r_expr_rel(++cont, ts);
513     a.meter(a.raiz, a2.raiz);
514
515     reg = ts.get(cont);
516     if (reg.TOKEN_ID != Tokens.T_RPAREN) {
517         PilaErrores.meter(new RegistroErr(230, reg.LINE, reg.NOMBRE));
518         throw new SecuenciaIncorrectaException();
519     }
520     actual = new NodoArbol();
521     actual.TOKEN_ID = Tokens.T_RPAREN;
522     actual.REF = reg.ID;
523     a.meter(a.raiz, actual);
524
525     a2 = r_bloque(++cont, ts);
526     a.meter(a.raiz, a2.raiz);
527
528     return a;
529 }
530 /**
531 * Analiza la estructura sintáctica de la lectura y regresa su árbol
532 *
533 * @param i el índice del primer token en la tabla
534 * @param ts la tabla de símbolos
535 * @return el árbol sintáctico de la estructura
536 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
537 */
538 public static ArbolSintactico r_lectura(int i, TablaSimbolos ts) throws
539 SecuenciaIncorrectaException {
540     cont = i;
541     ArbolSintactico a = new ArbolSintactico();
542     RegistroTS reg;
543     NodoArbol actual;
544     actual = new NodoArbol();
```

```
546     actual.REGLA_ID = ReglasProd.R_LECTURA;
547     a.raiz = actual;
548
549     reg = ts.get(cont);
550     actual = new NodoArbol();
551     actual.TOKEN_ID = Tokens.T_READ;
552     actual.REF = reg.ID;
553     a.meter(a.raiz, actual);
554
555     reg = ts.get(++cont);
556     if (reg.TOKEN_ID != Tokens.T_VAR) {
557         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
558         throw new SecuenciaIncorrectaException();
559     }
560     actual = new NodoArbol();
561     actual.TOKEN_ID = Tokens.T_VAR;
562     actual.REF = reg.ID;
563     a.meter(a.raiz, actual);
564
565     reg = ts.get(++cont);
566     if (reg.TOKEN_ID != Tokens.T_SEMICOLON) {
567         PilaErrores.meter(new RegistroErr(210, reg.LINE, reg.NOMBRE));
568         throw new SecuenciaIncorrectaException();
569     }
570     actual = new NodoArbol();
571     actual.TOKEN_ID = Tokens.T_SEMICOLON;
572     actual.REF = reg.ID;
573     a.meter(a.raiz, actual);
574     cont++;
575     return a;
576 }
577
578 /**
579 * Analiza la estructura sintáctica de la impresión y regresa su árbol
580 *
```

```
581     * @param i el índice del primer token en la tabla
582     * @param ts la tabla de símbolos
583     * @return el árbol sintáctico de la estructura
584     * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
585     */
586    public static ArbolSintactico r_impresion(int i, TablaSimbolos ts) throws
587        SecuenciaIncorrectaException {
588        cont = i;
589        ArbolSintactico a = new ArbolSintactico();
590        RegistroTS reg;
591        NodoArbol actual;
592        actual = new NodoArbol();
593        actual.REGLA_ID = ReglasProd.R_IMPRESION;
594        a.raiz = actual;
595
596        reg = ts.get(cont);
597        actual = new NodoArbol();
598        actual.TOKEN_ID = reg.TOKEN_ID;
599        actual.REF = reg.ID;
600        a.meter(a.raiz, actual);
601
602        reg = ts.get(++cont);
603        switch (reg.TOKEN_ID) {
604            case Tokens.T_VAR:
605            case Tokens.T_STR_CONST:
606                actual = new NodoArbol();
607                actual.TOKEN_ID = reg.TOKEN_ID;
608                actual.REF = reg.ID;
609                a.meter(a.raiz, actual);
610                break;
611            default:
612                PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
613                throw new SecuenciaIncorrectaException();
614        }
```

```
615     reg = ts.get(++cont);
616     if (reg.TOKEN_ID != Tokens.T_SEMICOLON) {
617         PilaErrores.meter(new RegistroErr(210, reg.LINE, reg.NOMBRE));
618         throw new SecuenciaIncorrectaException();
619     } else {
620         actual = new NodoArbol();
621         actual.TOKEN_ID = reg.TOKEN_ID;
622         actual.REF = reg.ID;
623         a.meter(a.raiz, actual);
624     }
625     cont++;
626     return a;
627 }
628 }
629 /**
630 * Analiza la estructura sintáctica de los parametros y regresa su árbol
631 *
632 * @param i el índice del primer token en la tabla
633 * @param ts la tabla de símbolos
634 * @return el árbol sintáctico de la estructura
635 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
636 */
637 public static ArbolSintactico r_params(int i, TablaSimbolos ts) throws
638     SecuenciaIncorrectaException {
639     cont = i;
640     ArbolSintactico a = new ArbolSintactico();
641     RegistroTS reg;
642     NodoArbol actual;
643     actual = new NodoArbol();
644     actual.REGLA_ID = ReglasProd.R_PARAMS;
645     a.raiz = actual;
646
647     do {
648         reg = ts.get(cont);
```

```
649     switch (reg.TOKEN_ID) {
650         case Tokens.T_VAR:
651         case Tokens.T_INT_CONST:
652         case Tokens.T_REAL_CONST:
653         case Tokens.T_BOOL_CONST:
654         case Tokens.T_STR_CONST:
655             actual = new NodoArbol();
656             actual.TOKEN_ID = reg.TOKEN_ID;
657             actual.REF = reg.ID;
658             a.meter(a.raiz, actual);
659             break;
660         default:
661             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
662             throw new SecuenciaIncorrectaException();
663     }
664
665     reg = ts.get(++cont);
666     if (reg.TOKEN_ID == Tokens.T_COMMA) {
667         actual = new NodoArbol();
668         actual.TOKEN_ID = Tokens.T_COMMA;
669         actual.REF = reg.ID;
670         a.meter(a.raiz, actual);
671         cont++;
672     }
673     } while (reg.TOKEN_ID == Tokens.T_COMMA);
674     return a;
675 }
676
677 /**
678 * Analiza la estructura sintáctica de los argumentos y regresa su árbol
679 *
680 * @param i el índice del primer token en la tabla
681 * @param ts la tabla de símbolos
682 * @return el árbol sintáctico de la estructura
683 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
```

```
684     */
685     public static ArbolSintactico r_args(int i, TablaSimbolos ts) throws
686     SecuenciaIncorrectaException {
687         cont = i;
688         ArbolSintactico a = new ArbolSintactico();
689         RegistroTS reg;
690         NodoArbol actual;
691         actual = new NodoArbol();
692         actual.REGLA_ID = ReglasProd.R_ARGS;
693         a.raiz = actual;
694
695         do {
696             reg = ts.get(cont);
697             switch (reg.TOKEN_ID) {
698                 case Tokens.T_INT:
699                 case Tokens.T_REAL:
700                 case Tokens.T_BOOL:
701                 case Tokens.T_STRING:
702                     actual = new NodoArbol();
703                     actual.TOKEN_ID = reg.TOKEN_ID;
704                     actual.REF = reg.ID;
705                     a.meter(a.raiz, actual);
706                     break;
707                 default:
708                     PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
709                     throw new SecuenciaIncorrectaException();
710
711             reg = ts.get(++cont);
712             if (reg.TOKEN_ID != Tokens.T_VAR) {
713                 PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
714                 throw new SecuenciaIncorrectaException();
715             } else {
716                 actual = new NodoArbol();
717                 actual.TOKEN_ID = Tokens.T_VAR;
```

```
718         actual.REF = reg.ID;
719         a.meter(a.raiz, actual);
720     }
721
722     reg = ts.get(++cont);
723     if (reg.TOKEN_ID == Tokens.T_COMMA) {
724         actual = new NodoArbol();
725         actual.TOKEN_ID = Tokens.T_COMMA;
726         actual.REF = reg.ID;
727         a.meter(a.raiz, actual);
728         cont++;
729     }
730 } while (reg.TOKEN_ID == Tokens.T_COMMA);
731 return a;
732 }
733 }
734 /**
735 * Analiza la estructura sintáctica de la asignacion y regresa su árbol
736 *
737 * @param i el índice del primer token en la tabla
738 * @param ts la tabla de símbolos
739 * @return el árbol sintáctico de la estructura
740 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
741 */
743 public static ArbolSintactico r_asignacion(int i, TablaSimbolos ts) throws
744 SecuenciaIncorrectaException {
745     cont = i;
746     ArbolSintactico a = new ArbolSintactico();
747     RegistroTS reg;
748     NodoArbol actual;
749     actual = new NodoArbol();
750     actual.REGLA_ID = ReglasProd.R_ASIGNACION;
751     a.raiz = actual;
```

```
752     reg = ts.get(cont);  
753  
754     switch (reg.TOKEN_ID) {  
755         case Tokens.T_INT:  
756         case Tokens.T_REAL:  
757         case Tokens.T_BOOL:  
758         case Tokens.T_STRING:  
759             actual = new NodoArbol();  
760             actual.TOKEN_ID = reg.TOKEN_ID;  
761             actual.REF = reg.ID;  
762             a.meter(a.raiz, actual);  
763             reg = ts.get(++cont);  
764             break;  
765     }  
766  
767     if (reg.TOKEN_ID != Tokens.T_VAR) {  
768         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
769         throw new SecuenciaIncorrectaException();  
770     } else {  
771         actual = new NodoArbol();  
772         actual.TOKEN_ID = Tokens.T_VAR;  
773         actual.REF = reg.ID;  
774         a.meter(a.raiz, actual);  
775     }  
776  
777     reg = ts.get(++cont);  
778     if (reg.TOKEN_ID != Tokens.T_ASSIGN) {  
779         PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));  
780         throw new SecuenciaIncorrectaException();  
781     } else {  
782         actual = new NodoArbol();  
783         actual.TOKEN_ID = Tokens.T_ASSIGN;  
784         actual.REF = reg.ID;  
785         a.meter(a.raiz, actual);  
786     }
```

```
787     cont++;
788     reg = ts.get(cont + 1);
789     ArbolSintactico a2;
790     switch (reg.TOKEN_ID) {
791         case Tokens.T_PLUS:
792         case Tokens.T_MINUS:
793         case Tokens.T_STAR:
794         case Tokens.T_DIV:
795             a2 = r_expr_aritm(cont, ts);
796             a.meter(a.raiz, a2.raiz);
797             break;
798         case Tokens.T_EQ:
799         case Tokens.T_LT:
800         case Tokens.T_GT:
801         case Tokens.T_NE:
802         case Tokens.T_LTE:
803         case Tokens.T_GTE:
804             a2 = r_expr_rel(cont, ts);
805             a.meter(a.raiz, a2.raiz);
806             break;
807         default:
808             reg = ts.get(cont);
809             switch (reg.TOKEN_ID) {
810                 case Tokens.T_INT_CONST:
811                 case Tokens.T_REAL_CONST:
812                 case Tokens.T_BOOL_CONST:
813                 case Tokens.T_STR_CONST:
814                 case Tokens.T_VAR:
815                     actual = new NodoArbol();
816                     actual.TOKEN_ID = reg.TOKEN_ID;
817                     actual.REF = reg.ID;
818                     a.meter(a.raiz, actual);
819                     cont++;
820                     break;
```

```

822         case Tokens.T_FUN:
823             a2 = r_call(cont, ts);
824             a.meter(a.raiz, a2.raiz);
825             break;
826         }
827     }
828     reg = ts.get(cont);
829     if (reg.TOKEN_ID != Tokens.T_SEMICOLON) {
830         PilaErrores.meter(new RegistroErr(210, reg.LINE, reg.NOMBRE));
831         throw new SecuenciaIncorrectaException();
832     }
833     actual = new NodoArbol();
834     actual.TOKEN_ID = Tokens.T_SEMICOLON;
835     actual.REF = reg.ID;
836     a.meter(a.raiz, actual);
837     cont++;
838     return a;
839 }
840 /**
841 * Analiza la estructura sintáctica de la expresión relacional y regresa su árbol
842 *
843 * @param i el índice del primer token en la tabla
844 * @param ts la tabla de símbolos
845 * @return el árbol sintáctico de la estructura
846 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
847 * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
848 estructura dentro de un bloque.
849 */
850 public static ArbolSintactico r_expr_rel(int i, TablaSimbolos ts) throws
851 SecuenciaIncorrectaException {
852     cont = i;
853     ArbolSintactico a = new ArbolSintactico();
854     RegistroTS reg;

```

```
855     NodoArbol actual;
856     actual = new NodoArbol();
857     actual.REGLA_ID = ReglasProd.R_EXPR_REL;
858     a.raiz = actual;
859
860     reg = ts.get(cont);
861
862     switch (reg.TOKEN_ID) {
863         case Tokens.T_INT_CONST:
864         case Tokens.T_REAL_CONST:
865         case Tokens.T_VAR:
866         case Tokens.T_STR_CONST:
867             actual = new NodoArbol();
868             actual.TOKEN_ID = reg.TOKEN_ID;
869             actual.REF = reg.ID;
870             a.meter(a.raiz, actual);
871             break;
872         default:
873             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
874             throw new SecuenciaIncorrectaException();
875     }
876     reg = ts.get(++cont);
877
878     switch (reg.TOKEN_ID) {
879         case Tokens.T_GTE:
880         case Tokens.T_LTE:
881         case Tokens.T_GT:
882         case Tokens.T_LT:
883         case Tokens.T_EQ:
884         case Tokens.T_NE:
885             actual = new NodoArbol();
886             actual.TOKEN_ID = reg.TOKEN_ID;
887             actual.REF = reg.ID;
888             a.meter(a.raiz, actual);
889             break;
```

```
890         default:
891             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
892             throw new SecuenciaIncorrectaException();
893     }
894     reg = ts.get(++cont);
895
896     switch (reg.TOKEN_ID) {
897         case Tokens.T_INT_CONST:
898         case Tokens.T_REAL_CONST:
899         case Tokens.T_VAR:
900         case Tokens.T_STR_CONST:
901             actual = new NodoArbol();
902             actual.TOKEN_ID = reg.TOKEN_ID;
903             actual.REF = reg.ID;
904             a.meter(a.raiz, actual);
905             break;
906         default:
907             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
908             throw new SecuenciaIncorrectaException();
909     }
910     cont++;
911     return a;
912 }
913
914 /**
915 * Analiza la estructura sintáctica de la expresión aritmética y regresa su árbol
916 *
917 * @param i el índice del primer token en la tabla
918 * @param ts la tabla de símbolos
919 * @return el árbol sintáctico de la estructura
920 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
921 */
922 public static Arbolsintactico r_expr_aritm(int i, TablaSimbolos ts) throws
923 SecuenciaIncorrectaException {
924     cont = i;
```

```
924     ArbolSintactico a = new ArbolSintactico();
925     RegistroTS reg;
926     NodoArbol actual;
927     actual = new NodoArbol();
928     actual.REGLA_ID = ReglasProd.R_EXPR_ARITM;
929     a.raiz = actual;
930
931     reg = ts.get(cont);
932
933     switch (reg.TOKEN_ID) {
934         case Tokens.T_INT_CONST:
935         case Tokens.T_REAL_CONST:
936         case Tokens.T_VAR:
937         case Tokens.T_STR_CONST:
938             actual = new NodoArbol();
939             actual.TOKEN_ID = reg.TOKEN_ID;
940             actual.REF = reg.ID;
941             a.meter(a.raiz, actual);
942             break;
943         default:
944             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
945             throw new SecuenciaIncorrectaException();
946     }
947     reg = ts.get(++cont);
948
949     switch (reg.TOKEN_ID) {
950         case Tokens.T_PLUS:
951         case Tokens.T_MINUS:
952         case Tokens.T_STAR:
953         case Tokens.T_DIV:
954             actual = new NodoArbol();
955             actual.TOKEN_ID = reg.TOKEN_ID;
956             actual.REF = reg.ID;
957             a.meter(a.raiz, actual);
958             break;
```

```
959         default:
960             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
961             throw new SecuenciaIncorrectaException();
962     }
963     reg = ts.get(++cont);
964
965     switch (reg.TOKEN_ID) {
966         case Tokens.T_INT_CONST:
967         case Tokens.T_REAL_CONST:
968         case Tokens.T_VAR:
969         case Tokens.T_STR_CONST:
970             actual = new NodoArbol();
971             actual.TOKEN_ID = reg.TOKEN_ID;
972             actual.REF = reg.ID;
973             a.meter(a.raiz, actual);
974             break;
975         default:
976             PilaErrores.meter(new RegistroErr(240, reg.LINE, reg.NOMBRE, reg.TOKEN_ID));
977             throw new SecuenciaIncorrectaException();
978     }
979     cont++;
980     return a;
981 }
982
983
984 /**
985 * Obtiene la regla de produccion a la que pertenece, con base en el primer token
986 *
987 * @param i el índice del primer token en la tabla
988 * @param ts la tabla de símbolos
989 * @return el árbol sintáctico de la estructura
990 * @throws SecuenciaIncorrectaException cuando se encuentra un token inesperado en la secuencia.
991 * @throws EstructuraNoReconocidaException cuando no se puede identificar una sentencia o
992 estructura dentro de un bloque.
993 */
```

```
993     public static int ObtenerReglaProd(int i, TablaSimbolos ts) throws
994         EstructuraNoReconocidaException {
995             RegistroTS reg;
996             cont = i;
997             reg = ts.get(cont);
998
999             switch (reg.TOKEN_ID) {
1000                 case Tokens.T_FUN:
1001                     return ReglasProd.R_CALL;
1002                 case Tokens.T_FUNC:
1003                     return ReglasProd.R_METODO;
1004                 case Tokens.T_RETURN:
1005                     return ReglasProd.R_RETORNO;
1006                 case Tokens.T_IF:
1007                     return ReglasProd.R_IF;
1008                 case Tokens.T_FOR:
1009                     return ReglasProd.R_FOR;
1010                 case Tokens.T_WHILE:
1011                     return ReglasProd.R WHILE;
1012                 case Tokens.T_READ:
1013                     return ReglasProd.R LECTURA;
1014                 case Tokens.T_PRINT:
1015                     return ReglasProd.R_IMPRESION;
1016                 case Tokens.T_VAR:
1017                 case Tokens.T_REAL:
1018                 case Tokens.T_INT:
1019                 case Tokens.T_BOOL:
1020                 case Tokens.T_STRING:
1021                     return ReglasProd.R_ASIGNACION;
1022             }
1023             throw new EstructuraNoReconocidaException();
1024         }
1025 }
```

```
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     private 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)
30                         );
31                         break;
32                     case ReglasProd.R_METODO:
33                         arboles.add(
34                             Estructuras.r_metodo(Estructuras.cont, ts)
35                         );
36                 }
37             }
38             while (Estructuras.cont < Estructuras.nroReglas);
39         }
40         catch (Error e) {
41             System.out.println("Error en la ejecución del análisis sintáctico: " + e.getMessage());
42         }
43         return true;
44     }
45 }
```

```
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     }
```

```
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 * Se encarga de imprimir los errores obtenidos del análisis sintáctico y errores que se
88 * presentaron en esta fase.
89 */
90 public void imprimirErrores() {
91     if (PilaErrores.size() > 0) {
92         System.err.println("Se encontraron errores durante el análisis sintáctico:");
93         while (PilaErrores.size() > 0) {
94             RegistroErr reg = PilaErrores.sacar();
95             Error err = BaseErrores.getError(reg.ERR_ID);
96             switch (reg.ERR_ID) {
97                 // De sistema
98                 case 12:
99                     err.setReason("Verifique que todas las llaves estén cerradas.");
100                    break;
101                // Sintácticos
102                case 210:
103                    err.setReason(String.format("(línea %d)", reg.LINEA_N));
104                    break;
105                case 220:
```

```
105             err.setReason(String.format("(línea %d)", reg.LINEA_N));
106             break;
107         case 230:
108             err.setReason(String.format("(línea %d)", reg.LINEA_N));
109             break;
110         case 240:
111             err.setReason(String.format("No se esperaba el token %s ('%s') en la línea
112 %d.", Tokens.nombres[reg.TOKEN_ID], reg.LEXEMA, reg.LINEA_N));
113             break;
114         case 250:
115             err.setReason(String.format("(línea %d)", reg.LINEA_N));
116             break;
117         }
118     System.err.println(err);
119 } else {
120     System.out.println("No se encontraron errores durante el análisis sintáctico");
121 }
122 }
123 /**
124 * Imprime los arboles generados despues de la validacion de la secuencia de los tokens
125 */
126 public void imprimirArboles() {
127     if (arboles != null)
128         for (ArbolSintactico arbol : arboles) {
129             System.out.println(arbol.toString());
130             System.out.println();
131         }
132     }
133 }
134
135 public boolean tieneArboles() {
136     return arboles != null && arboles.size() > 0;
137 }
138 }
```

```
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 }
```

A3 - EstructuraNoReconocidaException.java

```
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 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  * 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

Código acumulado hasta esta fase

A3 - Main.java

```
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 }
```

A3 - 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
45      setDefaultCloseOperation(javax.swing.WindowConstants.DO NOTHING ON CLOSE);
46      setTitle("E1");
47      setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
48      setMinimumSize(new java.awt.Dimension(480, 360));
49      setPreferredSize(new java.awt.Dimension(800, 600));
50
51      jToolBar1.setFloatable(false);
52      jToolBar1.setRollover(true);
53      jToolBar1.setDoubleBuffered(true);
54
55      btnLex.setText("Léxico");
56      btnLex.setToolTipText("Análisis léxico (F5)");
57      btnLex.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.
createEtchedBorder(), null));
58      btnLex.setDoubleBuffered(true);
59      btnLex.setFocusable(false);
60      btnLex.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
61      btnLex.setMaximumSize(new java.awt.Dimension(80, 28));
62      btnLex.setMinimumSize(new java.awt.Dimension(80, 28));
63      btnLex.setPreferredSize(new java.awt.Dimension(80, 28));
64      btnLex.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
65      jToolBar1.add(btnLex);
66
67      btnSyn.setText("Sintáctico");
68      btnSyn.setBorder(javax.swing.BorderFactory.createCompoundBorder(javax.swing.BorderFactory.
```

A3 - MainFrame.java

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

```
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.setDividerLocation(350);
126     jSplitPane.setOrientation(javax.swing.JSplitPane.VERTICAL_SPLIT);
127     jSplitPane.setResizeWeight(1.0);
128     jSplitPane.setAutoscrolls(true);
129     jSplitPane.setCursor(new java.awt.Cursor(java.awt.Cursor.N_RESIZE_CURSOR));
130     jSplitPane.setDoubleBuffered(true);
131     jSplitPane.setName("");
132
133     rTextScrollPane1.setLineNumbersEnabled(true);
```

```
134  
135     rSyntaxTextArea1.setColumns(20);  
136     rSyntaxTextArea1.setRows(5);  
137     rSyntaxTextArea1.setCodeFoldingEnabled(true);  
138     rSyntaxTextArea1.setFadeCurrentLineHighlight(true);  
139     rSyntaxTextArea1.setFont(new java.awt.Font("Consolas", 0, 12)); // NOI18N  
140     rSyntaxTextArea1.setMarginLineEnabled(true);  
141     rSyntaxTextArea1.setMarkOccurrences(true);  
142     rSyntaxTextArea1.setPaintMatchedBracketPair(true);  
143     rSyntaxTextArea1.setPaintTabLines(true);  
144     rSyntaxTextArea1.setTabsEmulated(true);  
145     rTextScrollPane1.setViewportView(rSyntaxTextArea1);  
146  
147     jSplitPane.setTopComponent(rTextScrollPane1);  
148  
149     jScrollPane2.setAutoscrolls(true);  
150     jScrollPane2.setDoubleBuffered(true);  
151     jScrollPane2.setMaximumSize(new java.awt.Dimension(32767, 400));  
152     jScrollPane2.setMinimumSize(new java.awt.Dimension(20, 0));  
153  
154     txtOut.setEditable(false);  
155     txtOut.setFont(new java.awt.Font("Monospaced", 0, 12)); // NOI18N  
156     txtOut.setToolTipText("");  
157     txtOut.setDoubleBuffered(true);  
158     txtOut.setMaximumSize(new java.awt.Dimension(2147483647, 400));  
159     jScrollPane2.setViewportView(txtOut);  
160  
161     jSplitPane.setRightComponent(jScrollPane2);  
162  
163     getContentPane().add(jSplitPane, java.awt.BorderLayout.CENTER);  
164  
165     jMenuBar1.setDoubleBuffered(true);  
166  
167     jMenu1.setText("Archivo");  
168     jMenu1.setDoubleBuffered(true);
```

```
169      menuFileOpen.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_O
170 , java.awt.event.InputEvent.CTRL_MASK));
171      menuFileOpen.setText("Abrir");
172      menuFileOpen.setToolTipText("");
173      menuFileOpen.setDoubleBuffered(true);
174      jMenu1.add(menuFileOpen);
175
176      menuFileSave.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_S
177 , java.awt.event.InputEvent.CTRL_MASK));
178      menuFileSave.setText("Guardar");
179      menuFileSave.setDoubleBuffered(true);
180      jMenu1.add(menuFileSave);
181
182      menuFileSaveAs.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.
183 VK_S, java.awt.event.InputEvent.SHIFT_MASK | java.awt.event.InputEvent.CTRL_MASK));
184      menuFileSaveAs.setText("Guardar como");
185      menuFileSaveAs.setDoubleBuffered(true);
186      jMenu1.add(menuFileSaveAs);
187
188      jSeparator1.setDoubleBuffered(true);
189      jMenu1.add(jSeparator1);
190
191      menuExit.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_F4,
192 java.awt.event.InputEvent.ALT_MASK));
193      menuExit.setText("Salir");
194      menuExit.setToolTipText("");
195      menuExit.setDoubleBuffered(true);
196      jMenu1.add(menuExit);
197
198      jMenuBar1.add(jMenu1);
199
200      jMenu2.setText("Ayuda");
201
202      btnMenuAyudaLex.setText("Análisis léxico");
```

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

```
235
236     public JSplitPane getjSplitPane() {
237         return jSplitPane;
238     }
239
240     public JMenuItem getMenuExit() {
241         return menuExit;
242     }
243
244     public JMenuItem getMenuFileOpen() {
245         return menuFileOpen;
246     }
247
248     public JMenuItem getMenuFileSave() {
249         return menuFileSave;
250     }
251
252     public JMenuItem getMenuFileSaveAs() {
253         return menuFileSaveAs;
254     }
255
256     public JMenuItem getMenuAyudaLex() {
257         return btnMenuAyudaLex;
258     }
259
260     public JMenuItem getMenuAyudaSyn() {
261         return btnMenuAyudaSyn;
262     }
263
264     public JTextPane getTxtOut() {
265         return txtOut;
266     }
267
268     public RSyntaxTextArea getCode() {
269         return rSyntaxTextArea1;
```

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

```
305     @Override
306     public void insertString(int offs, String str, AttributeSet a) throws BadLocationException
307     {
308         str = str.replaceAll("\t", " ");
309         super.insertString(offs, str, a);
310     }
311 }
312
```

A3 - TablaSimbolosDialog.java

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

```
36     private AnalizadorSintactico as;
37
38     /**
39      * Constructor de la clase.
40      * Muestra la vista automáticamente.
41      */
42     public UIController() {
43         this.frm = new MainFrame();
44         this.dlgTable = new TablaSimbolosDialog();
45         this.fc = new JFileChooser();
46         this.al = new AnalizadorLexico();
47         this.as = new AnalizadorSintactico();
48         dlgTable.setLocationRelativeTo(frm);
49         initListeners();
50         configView();
51         setOutPaneVisible(false);
52         frm.setVisible(true);
53         digest = "";
54     }
55
56     /**
57      * Inicializa los listeners de la vista.
58      */
59     private void initListeners() {
60         // Frame
61         frm.addWindowListener(new WindowAdapter() {
62             @Override
63             public void windowClosing(WindowEvent e) {
64                 super.windowClosing(e);
65                 salir();
66             }
67         });
68
69         // Editor
70         frm.getTxtCode().addKeyListener(new KeyListener() {
```

```
71     @Override
72     public void keyTyped(KeyEvent e) {
73
74     }
75
76     @Override
77     public void keyPressed(KeyEvent e) {
78         switch (e.getKeyCode()) {
79             case KeyEvent.VK_F5:
80                 if (frm.getBtnLex().isEnabled()) {
81                     if ((e.getModifiers() & ActionEvent.SHIFT_MASK) > 0)
82                         all();
83                     else
84                         lex();
85                 }
86                 break;
87             case KeyEvent.VK_T:
88                 if (frm.getBtnSTable().isEnabled())
89                     if ((e.getModifiers() & ActionEvent.CTRL_MASK) > 0)
90                         toggleTSDialog();
91                 }
92
93         }
94
95     @Override
96     public void keyReleased(KeyEvent e) {
97
98     }
99 }) ;
100
101 // Menu archivo
102 frm.getMenuFileOpen().addActionListener(e -> abrirArchivo());
103 frm.getMenuFileSave().addActionListener(e -> guardarArchivo());
104 frm.getMenuFileSaveAs().addActionListener(e -> guardarComo());
105 frm.getMenuAyudaLex().addActionListener(e -> {
```

```
106     if (Desktop.isDesktopSupported()) {
107         try {
108             Desktop.getDesktop().open(new File("docs/lexico.pdf"));
109         } catch (IOException e1) {
110             e1.printStackTrace();
111         }
112     }
113 }
114 frm.getMenuAyudaSyn().addActionListener(e -> {
115     if (Desktop.isDesktopSupported()) {
116         try {
117             Desktop.getDesktop().open(new File("docs/sintactico.pdf"));
118         } catch (IOException e1) {
119             e1.printStackTrace();
120         }
121     }
122 })
123 frm.getMenuExit().addActionListener(e -> salir());
124
125 // Toolbar
126 frm.getBtnLex().addActionListener(e -> lex());
127 frm.getBtnSyn().addActionListener(e -> syn());
128 frm.getBtnSem().addActionListener(e -> sem());
129 frm.getBtnAll().addActionListener(e -> sem());
130 frm.getBtnSTable().addActionListener(e -> toggleTSDialog());
131 frm.getBtnOutPane().addActionListener(e -> toggleOutPane());
132 }
133
134 /**
135 * Configura los objetos de la vista.
136 */
137 private void configView() {
138     // Se redirigen las salidas al textpane
139     UIOutputStreamController out = new UIOutputStreamController(frm.getTxtOut());
140     UIErrorStreamController err = new UIErrorStreamController(frm.getTxtOut());
```

```
141     System.setOut(new PrintStream(out));
142     System.setErr(new PrintStream(err));
143
144     // Configuración del FileChooser
145     fc.setFileHidingEnabled(true);
146     fc.setAcceptAllFileFilterUsed(false);
147     fc.setMultiSelectionEnabled(false);
148     fc.setFileFilter(new FileNameExtensionFilter("Programa E1", "el", "E1"));
149
150     // Botones
151     frm.getMenuFileSave().setEnabled(false);
152     frm.getBtnLex().setEnabled(false);
153     frm.getBtnSyn().setEnabled(false);
154     frm.getBtnSem().setEnabled(false);
155     frm.getBtnAll().setEnabled(false);
156     frm.getBtnSTable().setEnabled(false);
157
158     //Editor
159     AbstractTokenMakerFactory atmf = (AbstractTokenMakerFactory) TokenMakerFactory.
getDefaultInstance();
160     atmf.putMapping("text/E1", "itc.automatas2.gui.lib.E1TokenMaker");
161     FoldParserManager.get().addFoldParserMapping("text/E1", new CurlyFoldParser());
162     frm.getTxtCode().setSyntaxEditingStyle("text/E1");
163     frm.getTxtCodeScrollPane().setLineNumbersEnabled(true);
164     frm.getTxtCode().setCodeFoldingEnabled(true);
165 }
166
167 /**
168 * Abre un archivo utilizando un {@link JFileChooser FileChooser}.
169 */
170 private void abrirArchivo() {
171     fc.setDialogTitle("Abrir");
172     fc.setCurrentDirectory(new File("."));
173     if (fc.showOpenDialog(frm) == JFileChooser.APPROVE_OPTION) {
174         File file = fc.getSelectedFile();
```

```
175     handle = new ArchivoModel(file);
176     SwingUtilities.invokeLater(() -> {
177         RSyntaxTextArea code = frm.getTxtCode();
178         String content = handle.readAll();
179         code.setText(content);
180         digest = Util.md5(code.getText());
181         code.discardAllEdits();
182         frm.getMenuFileSave().setEnabled(true);
183         updateTitle(handle.getFile());
184         frm.getBtnLex().setEnabled(true);
185         frm.getBtnSyn().setEnabled(false);
186         frm.getBtnAll().setEnabled(false);
187         frm.getBtnSTable().setEnabled(false);
188         vaciarYOcultarTS();
189     });
190 }
191 }
192
193 /**
194 * Guarda el archivo actual.
195 */
196 private void guardarArchivo() {
197     SwingUtilities.invokeLater(() -> {
198         String content = frm.getTxtCode().getText();
199         handle.write(content);
200         digest = Util.md5(content);
201         frm.getBtnSyn().setEnabled(false);
202     });
203 }
204
205 /**
206 * Guarda un archivo utilizando un {@link JFileChooser FileChooser}.
207 */
208 private void guardarComo() {
209     fc.setDialogTitle("Guardar como");
```

```
210         if (fc.showSaveDialog(frm) == JFileChooser.APPROVE_OPTION) {
211             String path = fc.getSelectedFile().toString().endsWith(".el")
212                 ? fc.getSelectedFile().toString()
213                 : fc.getSelectedFile().toString() + ".el";
214             File file = new File(path);
215             handle = new ArchivoModel(file);
216             SwingUtilities.invokeLater(() -> {
217                 if (codeChanged()) {
218                     String content = frm.getTxtCode().getText();
219                     handle.write(content);
220                     digest = Util.md5(content);
221                 }
222                 frm.getMenuFileSave().setEnabled(true);
223                 updateTitle(handle.getFile());
224                 frm.getBtnLex().setEnabled(true);
225                 frm.getBtnSyn().setEnabled(false);
226                 frm.getBtnAll().setEnabled(false);
227             });
228         }
229     }
230 }
231 /**
232 * Maneja el cierre del programa y los cambios al código.
233 */
234 private void salir() {
235     if (codeChanged()) {
236         int option = JOptionPane.showConfirmDialog(frm,
237             "¿Desea guardar los cambios?", "Salir",
238             JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.QUESTION_MESSAGE
239         );
240         switch (option) {
241             case JOptionPane.YES_OPTION:
242                 guardarArchivo();
243             case JOptionPane.NO_OPTION:
```

```
245             System.exit(0);
246             break;
247         case JOptionPane.CANCEL_OPTION:
248         case JOptionPane.CLOSED_OPTION:
249             System.out.println("Cancelled");
250             break;
251         }
252     } else System.exit(0);
253 }
254
255 /**
256 * Muestra u oculta el panel de la salida del programa.
257 *
258 * @param visible la visibilidad del panel.
259 */
260 private void setOutPaneVisible(boolean visible) {
261     SwingUtilities.invokeLater(() -> {
262         JSplitPane pane = frm.getjSplitPane();
263         if (!visible) {
264             pane.setDividerLocation(1.0d);
265             frm.getTxtOut().setText("");
266         } else {
267             pane.setDividerLocation(pane.getHeight() - 200);
268         }
269         pane.setEnabled(visible);
270     });
271 }
272
273 /**
274 * Alterna la visibilidad del panel de salida.
275 */
276 private void toggleOutPane() {
277     setOutPaneVisible(!frm.getjSplitPane().isEnabled());
278 }
```

```
280     /**
281      * Alerta al usuario que hubo cambios al código.
282     */
283     private void promptChanged() {
284         JOptionPane.showMessageDialog(frm,
285             "Los cambios deben ser guardados para proceder con el análisis.\n" +
286             "Por favor, guarde y vuelva a intentar.", "Advertencia",
287             JOptionPane.WARNING_MESSAGE);
288     }
289
290     /**
291      * Ejecuta el análisis léxico del programa actual.
292     */
293     private void lex() {
294         if (codeChanged()) {
295             promptChanged();
296         } else if (handle != null) {
297             SwingUtilities.invokeLater(() -> {
298                 if (!frm.getjSplitPane().isEnabled())
299                     setOutPaneVisible(true);
300                 System.out.printf("ANÁLISIS LÉXICO DEL PROGRAMA \"%s\"\n", handle.getFile().toString
301 ());
302                 if (al.analizar(handle.getFile().toString())) {
303                     System.out.println("El analizador léxico declaró el código como válido");
304                     frm.getBtnSyn().setEnabled(true);
305                 } else {
306                     System.err.println("El analizador léxico declaró el código como inválido");
307                     frm.getBtnSyn().setEnabled(false);
308                 }
309                 poblarTS();
310                 frm.getBtnSTable().setEnabled(true);
311                 al.imprimirErrores();
312                 System.out.println();
313             });
314         }
315     }
```

```
314     }
315
316     /**
317      * Ejecuta el análisis sintáctico del programa actual.
318     */
319     private void syn() {
320         if (codeChanged()) {
321             promptChanged();
322         } else if (al.tS.size() > 0) {
323             SwingUtilities.invokeLater(() -> {
324                 if (!frm.getjSplitPane().isEnabled())
325                     setOutPaneVisible(true);
326                 System.out.printf("ANÁLISIS SINTÁCTICO DEL PROGRAMA \"%s\"\n", handle.getFile().
327 toString());
328                 if (as.analizar(al.tS)) {
329                     System.out.println("El analizador sintáctico declaró el código como válido");
330                     //frm.getBtnSem().setEnabled(true);
331                 } else {
332                     System.err.println("El analizador sintáctico declaró el código como inválido");
333                     //frm.getBtnSem().setEnabled(false);
334                 }
335                 if (as.tieneArboles()) {
336                     System.out.println("Árboles construidos:");
337                     as.imprimirArboles();
338                 }
339                 as.imprimirErrores();
340                 System.out.println();
341             });
342         }
343
344     /**
345      * Ejecuta el análisis semántico del programa actual.
346     */
347     private void sem() {
```

```
348         System.err.println("I SLEEP");  
349     }  
350  
351     /**  
352      * Ejecuta todos los análisis en cadena.  
353      */  
354     private void all() {  
355         System.err.println("No implementado aun!");  
356     }  
357  
358     /**  
359      * Limpia la tabla de símbolos y oculta su ventana.  
360      */  
361     private void vaciarYOcultarTS() {  
362         SwingUtilities.invokeLater(() -> {  
363             DefaultTableModel dtm = (DefaultTableModel) dlgTable.getSymTable().getModel();  
364             dtm.setRowCount(0);  
365             dlgTable.setVisible(false);  
366         });  
367     }  
368  
369     /**  
370      * Llena la tabla de símbolos.  
371      */  
372     private void poblarTS() {  
373         SwingUtilities.invokeLater(() -> {  
374             DefaultTableModel dtm = (DefaultTableModel) dlgTable.getSymTable().getModel();  
375             dtm.setRowCount(0);  
376             al.tS.registros().forEach(reg -> dtm.addRow(new Object[] {  
377                 reg.ID,  
378                 reg.NOMBRE,  
379                 Tokens.nombres[reg.TOKEN_ID],  
380                 Tipos.nombres[reg.TIPO],  
381                 reg.VAL,  
382                 reg.LINE  
383             }));  
384         });  
385     }  
386 }
```

```
383         });
384         Util.autosizeColumns(dlgTable.getSymTable(), 72, 16);
385     });
386 }
387
388 /**
389 * Alterna la visibilidad de la tabla de símbolos.
390 */
391 private void toggleTSDialog() {
392     dlgTable.setVisible(!dlgTable.isVisible());
393 }
394
395 /**
396 * Actualiza el título de la ventana a partir del nombre del archivo activo.
397 *
398 * @param activeFile el archivo activo.
399 */
400 private void updateTitle(File activeFile) {
401     frm.setTitle(String.format("%s [%s]", "E1", activeFile));
402 }
403
404
405 /**
406 * Verifica si hubo cambios en el código.
407 *
408 * @return <code>true</code> si se hicieron cambios.
409 */
410 private boolean codeChanged() {
411     String newDigest = Util.md5(frm.getTxtCode().getText());
412     return !digest.equals(newDigest);
413 }
414
415 }
416 }
```

```
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 }
```

A3 - UIOutputStreamController.java

```
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
```

A3 - BaseErrores.java

```
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     /**
29      * Obtiene un objeto {@link Error} de acuerdo al ID proporcionado.
30      *
31      * @param ID El identificador del error.
32      * @return Un objeto {@link Error} si existe, <code>null</code> si no.
33      */
34 }
```

```
35     public static Error getError(int ID) {  
36         return tablaErrores.get(ID);  
37     }  
38 }
```

```
1 package itc.automatas2.lexico;
2
3 /**
4  * Catálogo de tipos de datos
5 */
6 public class Tipos {
7     public static final int NONE = 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        "NONE",
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 }
```

A3 - Tokenizador.java

```
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.NONE;
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.NONE, tok.archivo.
71                                     getNoLinea()));
72             found = true;
73             break;
74         case "else":
75             tS.meter(new RegistroTS(token, Tokens.T_ELSE, Tipos.NONE, tok.archivo.
76                                     getNoLinea()));
77             found = true;
78             break;
79         case "while":
80             tS.meter(new RegistroTS(token, Tokens.T WHILE, Tipos.NONE, tok.archivo.
81                                     getNoLinea()));
82             found = true;
83             break;
84         case "for":
85             tS.meter(new RegistroTS(token, Tokens.T FOR, Tipos.NONE, tok.archivo.
86                                     getNoLinea()));
87             found = true;
88             break;
89         case "in":
90             tS.meter(new RegistroTS(token, Tokens.T_IN, Tipos.NONE, tok.archivo.
91                                     getNoLinea()));
92             found = true;
93             break;
94         case "func":
95             tS.meter(new RegistroTS(token, Tokens.T FUNC, Tipos.NONE, 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.NONE, tok.archivo
102                                     .getNoLinea()));
103            found = true;
```

```
97             break;
98         case "true":
99             tS.meter(new RegistroTS(token, Tokens.T_TRUE, Tipos.NONE, tok.archivo.
getNoLinea()));
100            if (foundVar && foundAssign) {
101                tS.getByID(lastIdentRef).VAL = token;
102                foundVar = false;
103                foundAssign = false;
104            }
105            found = true;
106            break;
107        case "false":
108            tS.meter(new RegistroTS(token, Tokens.T_FALSE, Tipos.NONE, tok.archivo.
getNoLinea()));
109            if (foundVar && foundAssign) {
110                tS.getByID(lastIdentRef).VAL = token;
111                foundVar = false;
112                foundAssign = false;
113            }
114            found = true;
115            break;
116        case "null":
117            tS.meter(new RegistroTS(token, Tokens.T_NULL, Tipos.NONE, tok.archivo.
getNoLinea()));
118            found = true;
119            break;
120        case "int":
121            tS.meter(new RegistroTS(token, Tokens.T_INT, Tipos.NONE, tok.archivo.
getNoLinea()));
122            ultimoTipo = Tipos.INT;
123            foundFunc = false;
124            found = true;
125            break;
126        case "real":
127            tS.meter(new RegistroTS(token, Tokens.T_REAL, Tipos.NONE, 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.NONE, tok.archivo.  
134             .getNoLinea()));  
135         ultimoTipo = Tipos.BOOL;  
136         foundFunc = false;  
137         found = true;  
138         break;  
139     case "string":  
140         tS.meter(new RegistroTS(token, Tokens.T_STRING, Tipos.NONE, tok.archivo  
141             .getNoLinea()));  
142         ultimoTipo = Tipos.STR;  
143         foundFunc = false;  
144         found = true;  
145         break;  
146     case "print":  
147         tS.meter(new RegistroTS(token, Tokens.T_PRINT, Tipos.NONE, tok.archivo.  
148             .getNoLinea()));  
149         found = true;  
150         break;  
151     case "println":  
152         tS.meter(new RegistroTS(token, Tokens.T_PRINTLN, Tipos.NONE, tok.  
153             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.NONE, tok.archivo.
164                 getNoLinea()));
165                 found = true;
166                 break;
167             case ";":
168                 tS.meter(new RegistroTS(token, Tokens.T_SEMICOLON, Tipos.NONE, tok.
169                 archivo.getNoLinea()));
170                 found = true;
171                 break;
172             case "(":
173                 tS.meter(new RegistroTS(token, Tokens.T_LPAREN, Tipos.NONE, tok.archivo
174                 .getNoLinea()));
175                 found = true;
176                 break;
177             case ")":
178                 tS.meter(new RegistroTS(token, Tokens.T_RPAREN, Tipos.NONE, tok.archivo
179                 .getNoLinea()));
180                 found = true;
181                 break;
182             case "{":
183                 tS.meter(new RegistroTS(token, Tokens.T_LBRACE, Tipos.NONE, tok.archivo
184                 .getNoLinea()));
185                 found = true;
186                 break;
```

```
186         case "[":
187             tS.meter(new RegistroTS(token, Tokens.T_LBRACKET, Tipos.NONE, tok.
188             archivo.getNoLinea())));
189             found = true;
190             break;
191         case "]":
192             tS.meter(new RegistroTS(token, Tokens.T_RBRACKET, Tipos.NONE, tok.
193             archivo.getNoLinea())));
194             found = true;
195             break;
196         case "<=":
197             tS.meter(new RegistroTS(token, Tokens.T_LTE, Tipos.NONE, tok.archivo.
198             getNoLinea())));
199             found = true;
200             break;
201         case ">=":
202             tS.meter(new RegistroTS(token, Tokens.T_GTE, Tipos.NONE, tok.archivo.
203             getNoLinea())));
204             found = true;
205             break;
206         case "!=":
207             tS.meter(new RegistroTS(token, Tokens.T_NE, Tipos.NONE, tok.archivo.
208             getNoLinea())));
209             found = true;
210             break;
211         case "<":
212             tS.meter(new RegistroTS(token, Tokens.T_LT, Tipos.NONE, tok.archivo.
213             getNoLinea())));
214             found = true;
215             break;
216         case ">":
217             tS.meter(new RegistroTS(token, Tokens.T_GT, Tipos.NONE, tok.archivo.
218             getNoLinea())));
219             found = true;
220             break;
```

```
214         case "==" :
215             tS.meter(new RegistroTS(token, Tokens.T_EQ, Tipos.NONE, tok.archivo.
216                                     getNoLinea()));
217             found = true;
218             break;
219         case "=" :
220             tS.meter(new RegistroTS(token, Tokens.T_ASSIGN, Tipos.NONE, tok.archivo.
221                                     .getNoLinea()));
222             found = true;
223             break;
224         case "--" :
225             tS.meter(new RegistroTS(token, Tokens.T_DEC, Tipos.NONE, tok.archivo.
226                                     getNoLinea()));
227             found = true;
228             break;
229         case "++" :
230             tS.meter(new RegistroTS(token, Tokens.T_INC, Tipos.NONE, tok.archivo.
231                                     getNoLinea()));
232             found = true;
233             break;
234         case "!" :
235             tS.meter(new RegistroTS(token, Tokens.T_NOT, Tipos.NONE, tok.archivo.
236                                     getNoLinea()));
237             found = true;
238             break;
239         case "+" :
240             tS.meter(new RegistroTS(token, Tokens.T_PLUS, Tipos.NONE, tok.archivo.
241                                     getNoLinea()));
242             found = true;
243             break;
244         case "-" :
245             tS.meter(new RegistroTS(token, Tokens.T_MINUS, Tipos.NONE, tok.archivo.
246                                     getNoLinea()));
247             found = true;
248             break;
```

```
242             case "*":
243                 tS.meter(new RegistroTS(token, Tokens.T_STAR, Tipos.NONE, tok.archivo.
244                                         getNoLinea()));
245                 found = true;
246                 break;
247             case "/":
248                 tS.meter(new RegistroTS(token, Tokens.T_DIV, Tipos.NONE, tok.archivo.
249                                         getNoLinea()));
250                 found = true;
251                 break;
252             case ">":
253                 tS.meter(new RegistroTS(token, Tokens.T_RANGE, Tipos.NONE, tok.archivo.
254                                         getNoLinea()));
255                 found = true;
256                 break;
257             case ",":
258                 tS.meter(new RegistroTS(token, Tokens.T_COMMA, Tipos.NONE, 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.NONE;
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.NONE,
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 }
```

```
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 pila de errores en la salida estándar.
362 */
363 public void imprimirErrores() {
364     if (PilaErrores.size() > 0) {
365         System.err.println("Se encontraron errores durante el análisis léxico:");
366         while (PilaErrores.size() > 0) {
367             RegistroErr reg = PilaErrores.sacar();
368             Error err = BaseErrores.getError(reg.ERR_ID);
369             switch (reg.ERR_ID) {
370                 //Genéricos
```

```

372             case 10:
373                 err.setReason(String.format("(RUTA: %s)", reg.LEXEMA));
374                 break;
375             case 11:
376                 err.setReason(String.format("(RUTA: %s)", reg.LEXEMA));
377                 break;
378             //Léxicos
379             case 110:
380                 err.setReason(String.format("No se pudo identificar el token \"%s\" en la
línea %d", reg.LEXEMA, reg.LINEA_N));
381                 break;
382             case 120:
383                 err.setReason(String.format("El token '%s' en la línea %d contiene símbolos
no reconocibles (fuera del código ASCII). ", reg.LEXEMA, reg.LINEA_N));
384                 break;
385             }
386             System.err.println(err);
387         }
388     } else {
389         System.out.println("No se encontraron errores durante el análisis léxico");
390     }
391 }
392 /**
393 * Imprime la tabla de símbolos en la salida estándar.
394 */
395 public void imprimirSimbolos() {
396     if (tS != null && tS.size() > 0) {
397         System.out.println(
398             "-----");
399         System.out.println(" | ");
400         System.out.println(
401             "-----");

```

TABLA DE SÍMBOLOS

```
400 -----") ;
401         System.out.printf("| %24s | %-30s | %-12s | %-12s | %24s |\n", "ID", "NOMBRE O LEXEMA",
402             "TOKEN ID", "TIPO DE DATO", "VALOR");
403         System.out.println(
404             "-----");
405         for (RegistroTS reg : tS.registros()) {
406             System.out.printf("| %24s | %-30s | %-12s | %-12s | %24s |\n", reg.ID, reg.NOMBRE,
407                 Tokens.nombres[reg.TOKEN_ID], Tipos.nombres[reg.TIPO], reg.VAL);
408         }
409         System.out.println(
410             "-----");
411     } else {
412         System.out.println("No se encontraron símbolos en el código");
413     }
414 }
```

```
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 }
```

A3 - AutomataIdentificador.java

```
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.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 }
```

A3 - RegistroTS.java

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

```
36      }
37 }
```

```
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
```

A3 - TablaSimbolos.java

```
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 }
```