

**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



TECNOLÓGICO NACIONAL DE MÉXICO  
en Celaya

**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 ANÁLISIS SINTÁCTICO ?
  - D. INVESTIGAR Y PROPOSER. ¿ 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.



**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: [inco@itcelaya.edu.mx](mailto:inco@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)



**ANAB**  
ASOCIACIÓN NACIONAL  
DE ASESORES Y CERTIFICADORES  
PROFESSIONAL



SECRETARÍA DE  
EDUCACIÓN PÚBLICA



TECNOLOGICO NACIONAL DE MEXICO  
en Celaya

3. COMO EQUIPO Y DERIVADO DEL ANÁLISIS ANTERIOR SE DEBEN PROPOSER 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 CORRELACIONADOS 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, ASÍ 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.



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](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)



**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



TECNOLOGICO NACIONAL DE MÉXICO  
en Celaya

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

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

Handwritten signature in black ink.

Handwritten signature in black ink.

Handwritten signature in blue ink.



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](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)



**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



TECNOLOGICO NACIONAL DE MEXICO  
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*).



**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](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)



**ANAB**  
ACCREDITED  
MANAGEMENT SYSTEMS  
CERTIFICATION BODY

SEP

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



TECNOLOGICO NACIONAL DE MEXICO  
en Celaya

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.



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](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)



**SEP**

SECRETARÍA DE  
EDUCACIÓN PÚBLICA



TECNOLOGICO NACIONAL DE MEXICO  
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 CONSECUATIVO POR EL QUE CORRESPONDA)
EQUIPO	: NÚMERO DEL EQUIPO QUE CORRESPONDA SEGÚN INDICACIÓN DEL PROFESOR.
NOCTROL	: SU NÚMERO DE CONTROL
APELLIDOS	: SUS APELLIDOS
NOMBRE	: SU NOMBRE
SEM	: EL PERIODO 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	M	J	V	S	D	L	M	M	J	V	S	D
			1	2	3	4				1			
3	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
							30						

EVALUACIÓN FORMATIVA DE 1<sup>RA</sup>OPORTUNIDAD (PARCIALES)



60 Aniversario  
TecNM  
en Celaya

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](mailto:lince@itcelaya.edu.mx)  
[www.itcelaya.edu.mx](http://www.itcelaya.edu.mx)





# INSTITUTO TECNOLÓGICO DE CELAYA

SEGUNDA ETAPA DE UN  
COMPILADOR:  
ANALISIS SINTACTICO

EQUIPO:  
CASTRO GONZALEZ RICARDO ISAAC  
CERVANTES VALADEZ ARCADIO  
HERNANDEZ ALTAMIRA LUIS FELIPE  
PANTOJA VALLE LAURA JAZMIN

CARRERA	MATERIA
INGENIERÍA EN SISTEMAS COMPUTACIONALES	LENGUAJES Y AUTOMATAS II

## 1 INTRODUCCIÓN

El analizador sintáctico es la segunda fase de un compilador, se encarga de verificar la secuencia de tokens que representa al texto de entrada, esto en base a una gramática dada. En caso de que la entrada sea válida, este proporciona el árbol sintáctico.

## 2 OBJETIVO (COMPETENCIA)

Proponer, diseñar e implementar las herramientas necesarias para el desarrollo de un analizador sintáctico, es decir, diagramas, estructuras de datos y programas para la construcción de la segunda etapa de un compilador.

## 3 FUNDAMENTO

LEA CUIDADOSAMENTE, Y REALICE LAS SIGUIENTES ACTIVIDADES, CONSIDERANDO LOS CRITERIOS DE CALIDAD PROPUESTOS EN LOS DOCUMENTOS DE LA [GUIA TUTORIAL](#), Y LA [RÚBRICA DE EVALUACIÓN](#).

1. TOMANDO EN CUENTA QUE LA PRIMERA ETAPA DEL ANALIZADOR 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.**
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?**

En la actividad anterior se desarrolló la fase de análisis léxico en la cual se extraía del archivo fuente todas las cadenas de caracteres que reconocía como parte del vocabulario y se generaba

un conjunto de tokens como salida y en caso de que parte del archivo de entrada no pudiera reconocerse como lenguaje valido se generaban los mensajes de error correspondientes, cabe mencionar que si el programa fuente tenía espacios en blanco o líneas en blanco así como comentarios o información que no era precisamente código se eliminaba del programa fuente.

Ahora bien en el análisis sintáctico se procesa la secuencia de tokens generada con anterioridad y los caracteres se agrupan jerárquicamente en frases gramaticales que el compilador utiliza para sintetizar una salida es decir se comprueba si es sintácticamente correcto esta es una representación intermedia aun no es lenguaje maquina pero le permitirá al compilador realizar su labor con más facilidad en las fases sucesivas por lo cual las frases gramaticales del programa fuente son expresadas mediante un árbol de análisis sintáctico.

Un análisis sintáctico es la fase que se encarga de checar el texto de entrada en base a una gramática ya definida previamente, y en caso de que esta entrada sea válida dentro del contexto, se suministra el árbol sintáctico que la reconoce.

A continuación, se muestra de manera gráfica como tanto la fase de análisis léxico como sintáctico están estrechamente relacionadas en la imagen podemos ver como a partir de un programa fuente comienza la etapa del analizador léxico y el uso de la tabla de símbolos y estos componentes léxicos son enviados a la siguiente fase que es el analizador sintáctico el cual requiere cada componente léxico para seguir analizándolo y en base a esto se suministra el árbol de análisis sintáctico que lo reconoce, pero la tabla de símbolos y el manejo de errores siguen en funcionamiento.

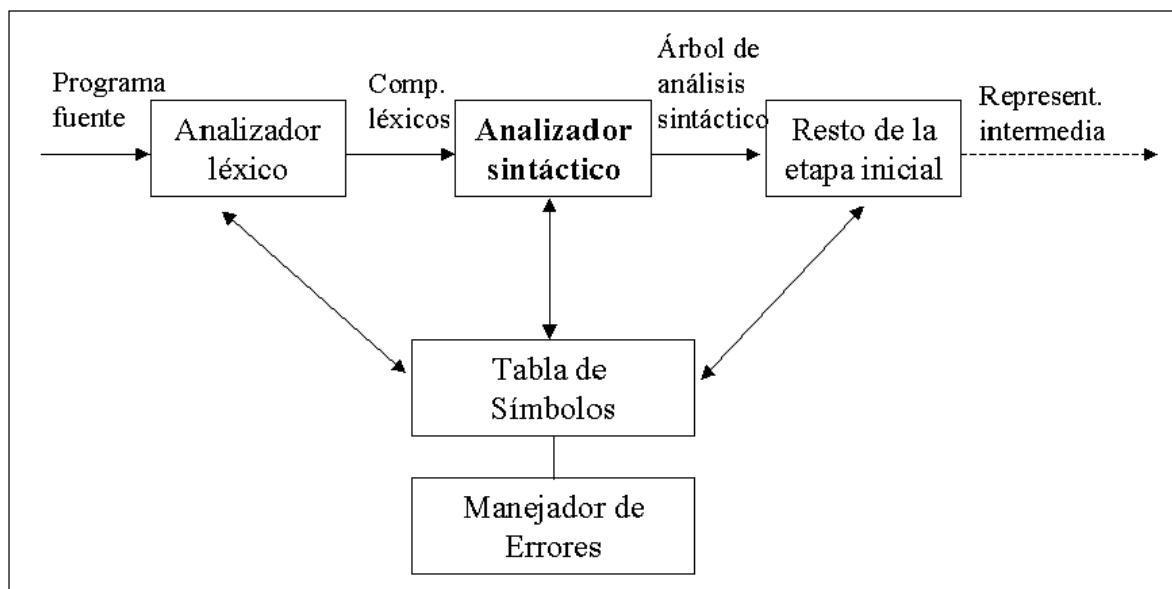


Imagen 1.- El analizador léxico en el proceso de la compilación

Entonces las funciones que debe de desempeñar un analizador sintáctico son:

- Acceder a la tabla de símbolos (para hacer parte del trabajo del analizador semántico)
- Chequeo de tipos (del analizador semántico)
- Generar errores cuando se producen.

## Manejo de errores sintácticos

El diseño y la implementación de un compilador sería muy sencillo si únicamente se procesaran programas que fueran libres de errores pero a menudo y lógicamente los programadores cometen errores y un buen compilador debe de indicar y localizar en qué parte fue en donde el programador cometió el error.

Los errores sintácticos incluyen la colocación incorrecta de los signos de **punto y coma**, además de llaves adicionales o faltantes; es decir “{” o “}”. Como otro ejemplo si tenemos una instrucción “**else**” sin una instrucción “**if**” esto genera un error sintáctico.

### Tipos de errores sintácticos

- **Fin inadecuado de instrucciones**

Por ejemplo dentro de nuestro lenguaje prototipo:

Es correcto poner la siguiente instrucción

`print("hola mundo");`

pero si se omite el ; al final de la instrucción sería un tipo de error de sintaxis

**`print("hola mundo")`**

- **Uso de paréntesis y llaves no equilibrados**

Ejemplo de uso llaves equilibradas:

`init (){`

`}`

Ejemplo de uso de llaves no equilibradas

`init (){`

Ejemplo de uso de paréntesis equilibrados

`init() {`

`}`

Ejemplo de uso de paréntesis no equilibrados

`Init ({`

`}`

El manejo de errores de sintaxis suele ser el más complicado porque al momento de encontrar un error el compilador debe de recuperarse y seguir buscando es decir el manejo de errores de un analizador sintáctico tiene que indicar de manera exacta el tipo de error y en donde se localizó, tiene que recuperarse para poder seguir buscando errores en el programa fuente y estas tareas no deben de hacer que el compilador entre en un estado de lentitud.

## B. INVESTIGAR. ¿EN QUÉ CONSISTE UN ANÁLISIS SINTÁCTICO Y QUÉ LO CARACTERIZA?

El análisis sintáctico consiste en tomar el programa fuente en forma de tokens previamente generados en el análisis léxico, este determina la estructura de las sentencias del programa.

En dado caso que el programa sea válido, proporciona el árbol sintáctico que lo reconoce.

A este proceso de reconocimiento del lenguaje fuente se le conoce también como **parsing**.

El analizador sintáctico agrupa a los tokens en clases sintácticas (se denominan **no terminales** en la definición de la gramática), como pueden ser expresiones, procedimientos, etc.

Este análisis obtiene un árbol sintáctico (u otra estructura equivalente) en la cual las hojas son representadas por los tokens, y cualquier nodo que no sea una hoja, se representa por un tipo de clase sintáctica (operaciones).

### Ejemplo del análisis sintáctico de una expresión:

$$(A+B)*(C+D)$$

`<expresión> :: = <termino> <más términos>`

`<más términos> :: = + <termino> <más términos> | = <termino> <más términos> | <vacío>`

`<termino> :: = <factor> <más factores>`

`<más factores> :: * <factor> <más factores> | / <factor> <más factores> | <vacío>`

`<factor> :: = (<expresión>) | <variable> | <constante>`

Esta estructura de la gramática refleja la prioridad que tienen los operadores, en este caso “+” y “-“ tienen la prioridad más baja, mientras que “\*” y “/” tienen una prioridad superior, de tal modo las constantes, variables y expresiones entre paréntesis se evaluarán en primer lugar. [1]

Los árboles sintácticos se construyen a través de un conjunto de reglas que son conocidas como **gramática**.

La principal función del analizador sintáctico no es verificar como tal que la sintaxis del programa fuente sea la adecuada, si no, construir una representación de manera interna de dicho programa con el fin de que si el programa es incorrecto mandar un mensaje de error.

Durante este proceso, el analizador sintáctico comprueba que el orden en que el analizador léxico va entregando los tokens es válido, si esto es correcto significa que la sucesión de símbolos que representa a los tokens puede ser generada por dicha gramática del programa fuente.

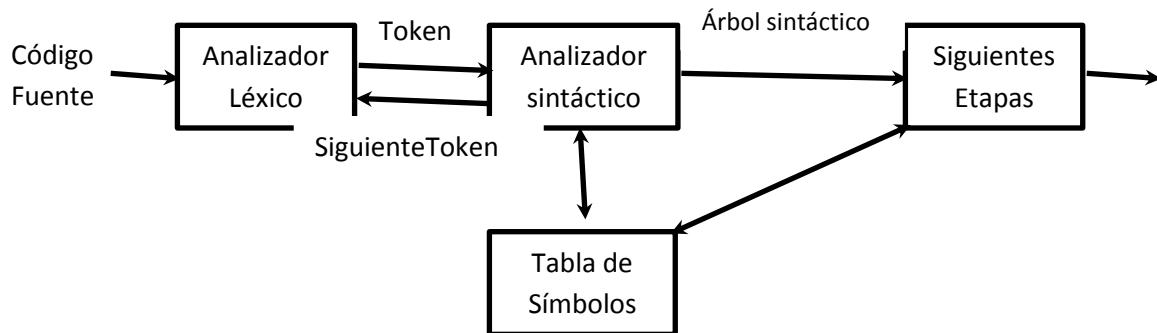
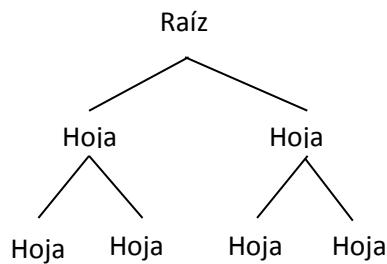


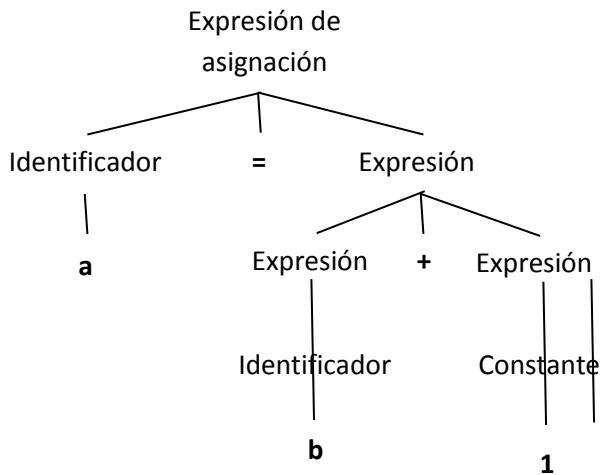
Imagen 2.B.1 Modelo del compilador

### Árboles de Análisis Sintáctico.

Un árbol de análisis sintáctico es la representación gráfica. Cada nodo interior de un árbol de análisis sintáctico representa la aplicación de una producción. El nodo interior se etiqueta con un **no terminal** en el encabezado de la producción; los hijos de este nodo se etiquetan de izquierda a derecha. Las hojas del árbol sintáctico se etiquetan con **no terminales** o **terminales** y se leen de izquierda a derecha, de esta manera se puede construir una “especie” de frase, a la cual le llamaremos **producto o frontera del árbol**.



**Imagen 2.B.2 Representación del Árbol Sintáctico**



**Imagen 2.B.3 Árbol Sintáctico de la expresión  $a=b+1$**

### Las principales funciones:

- Generar las estructuras de datos (árbol sintáctico u otras estructuras)
- Identificar cada tipo de instrucciones y los componentes correspondientes.
- Completar la tabla de símbolos.
- Validar las declaraciones de identificadores: es decir no se puede usar una variable si no ha sido declarada anteriormente.

### Categorías generales para los analizadores sintácticos:

- **Análisis descendente:** En este análisis las entradas parten de izquierda a derecha.

El análisis sintáctico descendente tiene como objetivo encontrar entre las producciones de la gramática la derivación por la izquierda del símbolo inicial para una cadena de entrada.

Este proceso parte del axioma de la gramática, enseguida se procesa la entrada de izquierda a derecha y por último escoge las reglas gramaticales.

Para poder trabajar el análisis sintáctico descendente se debe tener en cuenta realizar algunas operaciones para que la gramática sea **LL1** (**L: método direccional, procesamos la entrada**

**de izquierda a derecha (from left to right) L: obtenemos derivación más a la izquierda (left-most derivation) y 1: usamos un símbolo de preanálisis para decidir la producción a aplicar) las cuales son:**

- **Eliminar ambigüedad:** Para eliminar la ambigüedad se debe reescribir la gramática.
  - **Eliminar recursividad por la izquierda:** Una gramática es recursiva por la izquierda si tiene un nodo terminal, de tal forma que existe una derivación para alguna cadena. Es decir, por simple observación podemos identificar.
  - **Factorizar:** Reestructurar las producciones de la gramática con un igual comienzo, con el fin de analizar la entrada para elegir una opción correcta.
- 
- **Análisis ascendente:** Se comienza a construir el árbol por las hojas (donde se encuentran ubicados los tokens), se crean nodos intermedios hasta llegar a la raíz (símbolo inicial), así, se construye el árbol de abajo hacia arriba. El recorrido se hará desde las hojas hasta la raíz. El orden en el cual se encuentran las producciones corresponde a la inversa de una derivación por la derecha. Esto disminuye el número de reglas mal aplicadas con respecto al caso del análisis descendente.

#### Catálogo de Errores Sintácticos

DESCRIPCIÓN
Se esperaba ;
Se esperaba ID o INT o STRING
Se esperaba (
Se esperaba )
Se esperaba {
Se esperaba }

Ejemplos de estos

String = “10”;

En este claro ejemplo se espera un IDENTIFICADOR dado que en el BNF se necesita después de tipo de dato un IDENTIFICADOR

Otro ejemplo claro de este tipo de error sería

String \$id = ;

donde faltaría asignarle otro tipo de dato, se esperaría que se le asignará.

= 10;

en este caso se le faltaría un Identificador para que sea válido.

En una expresión booleana o matemática estaría dado un error de la siguiente manera

10 +

Generaría un error como el siguiente

“Se esperaba ID | INT | STRING”

En el caso de un booleana seria

10 <

lo cual generaría un error similar  
“Se esperaba ID | INT | STRING”

En estructuras de control se pueden dar los siguiente errores

“Se esperaba {“  
“Se esperaba }“  
“Se esperaba (“  
“Se esperaba )“

Esto se dan en las estructuras de control como while o if

## **INVESTIGAR Y PROPONER. ¿QUÉ PROCESOS Y PROBLEMAS ATIENDE UN ANÁLISIS SINTÁCTICO?**

### **Manejo de errores:**

Si un compilador tuviera que procesar solo programas correctos, su diseño e implementación serían demasiado simplificadas. Resulta evidente que los errores de corrección no pueden ser detectados por un compilador, ya que en ellos interviene el concepto abstracto que tiene del programador sobre el programa que está desarrollando.

El manejo de errores sintácticos es el más complicado desde el punto de vista del desarrollador del compilador. Nos interesa que cuando el compilador encuentre un error, este no se detenga, si no que siga su camino creando el árbol sintáctico, por lo tanto, debe de tener:

- Indicar los errores de forma clara y precisa.
- Recuperarse del error encontrado, para poder seguir examinando la entrada.
- Distinguir entre errores y advertencias. Las advertencias se suelen utilizar para informar sobre sentencias válidas, pero que estas crean un error lógico.
- No ralentizar significativamente la compilación.

### **Ignorar el problema:**

Esta estrategia o proceso consiste en ignorar el resto de la entrada, hasta poder encontrar una condición especial, que podría ser un token especial, (delimitador, un salto de línea, etc.). Y a partir de este punto, se sigue analizando normalmente. Todos los tokens encontrados desde el error hasta la condición especial, son descartados.

### **Recuperación a nivel de frase:**

Intentar corregir el error una vez encontrado, pero se debe de tener cuidado al implementar este proceso, ya que puede dar lugar a recuperaciones infinitas, esto es, situaciones en las que el intento de corrección no es el acertado, sino que introduce un nuevo error, que a su vez, se intenta corregir de la misma manera, y esto generando un ciclo de correcciones erróneas.

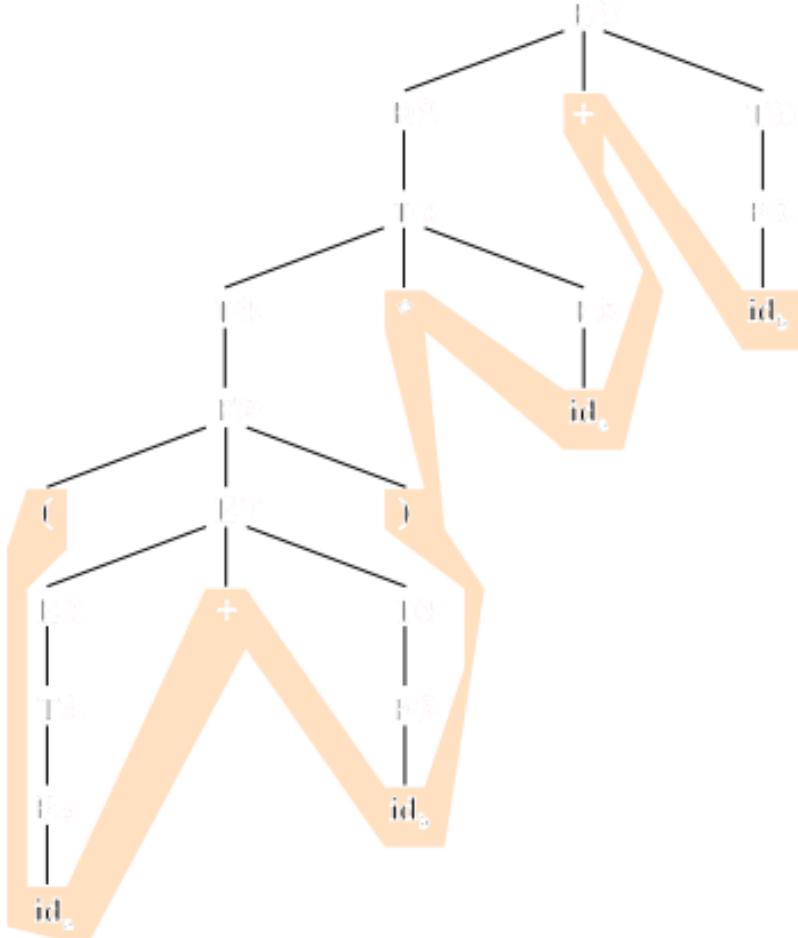
### **Creación del árbol sintáctico:**

Un árbol sintáctico se corresponde con una sentencia, obedece a una gramática y constituye una representación que se utiliza para describir el proceso de derivación de dicha sentencia, la raíz del árbol es el axioma inicial y, según nos convenga, lo dibujaremos en la cima o en el fondo del árbol.

Como nodos internos del árbol, se sitúan los elementos no terminales de las reglas de producción, y cada uno de ellos contendrá tantos hijos como símbolos existan en la parte derecha de la regla aplicada.

### **Ejemplo:**

“Veamos un ejemplo utilizando la gramática del cuadro 3.1; supongamos que hay que reconocer la cadena  $(a + b)^* a + b$  que el analizador léxico nos ha suministrado como  $(ida + idb)^* ida + idb$  y vamos a construir el árbol sintáctico, que sería el de la figura 3.2.”



**Figure 2** Árbol sintáctico correspondiente al reconocimiento de la secuencia  $(id_a + id_b)^* id_a + id_b$ , que se halla marcada de color.

La construcción de este árbol conlleva la aplicación inteligente de las reglas de producción. Si la sentencia a reconocer es incorrecta, esto es, no pertenece al lenguaje generado por la gramática, será imposible obtener su árbol sintáctico. En otras ocasiones una sentencia admite más de un árbol sintáctico. Cuando esto ocurre se dice que la gramática es ambigua. Este tipo de situaciones hay que evitarlas, de suerte que, o bien se cambia la gramática.

## E. INVESTIGAR. ¿CÓMO IMPLEMENTAR UN ANÁLISIS SINTÁCTICO?

Para implementar un análisis sintáctico existen **dos** categorías generales de algoritmos para la construcción: los analizadores sintáticos **descendentes** y los analizadores sintáticos **ascendentes**.

Estos a su vez, se dividen en varios métodos, los cuales tienen capacidades y propiedades diferentes.

### ANALIZADOR SINTÁCTICO DESCENDENTE (TOP-DOWN)

Este analizador construye el árbol sintáctico desde la raíz o el axioma hacia abajo. Estos se clasifican de la siguiente manera:

- **Descenso recursivo (retroceso).**

Estos contienen un conjunto de procedimientos recursivos, donde cada **no terminal** de dicha gramática tiene un procedimiento asociado. Cuando el árbol se ha construido en su totalidad, al momento de llegar al terminal, esta comprueba si el procedimiento es el adecuado y en caso de que lo sea, avanza al siguiente token, si el procedimiento es el incorrecto, este emite un error y aplica una estrategia de recuperación que pueda deshacer dicha operación, retroceder y llamar a otro procedimiento.

- **Predictivo LL(1).**

Las entradas son de izquierda a derecha, mientras que las construcciones de derivaciones por la izquierda de una sentencia o enunciado. La gramática que es analizada por este método es la gramática LL.

La gramática LL es demasiado restrictiva, pero los analizadores LL solo necesitan ver el siguiente token para hacer el análisis de sus decisiones.

El análisis consiste en la siguiente manera:

- Recibe un buffer de entrada y una cadena de gramática.
- Una pila para almacenar los símbolos terminales y los no terminales de la gramática aun sin analizar.
- Una tabla de análisis.

Este proceso analiza la sintaxis de dicha cadena de entrada, lo cual genera una derivación por la izquierda. Se puede decir que actúa como un programa obtenido de la traducción directa del autómata.

Aunque estos analizadores tienen el problema del no determinismo al momento de elegir entre dos posibles formas de reescribir el mismo no terminal y cuentan con solo una información.

Estas opciones se proporcionan en la gramática que debe generarse en los lenguajes que contienen más de una cadena. Es por eso que la actividad principal de los analizadores sintácticos LL es poder predecir cuál de las distintas reglas de escritura se debe usar para procesar los símbolos de entrada restante, es por eso que reciben el nombre de analizadores sintácticos predictivos.

### **ANALIZADOR SINTÁCTICO ASCENDENTE (BOTTOM-UP)**

Este analizador construye el árbol sintáctico de abajo hacia arriba.

- **Precedencia de operador.**

Este es utilizado sólo para conjuntos pequeños de gramáticas, denominados gramáticas de operadores. Cuenta con una propiedad, esta gramática no puede tener dos terminales seguidos.

- **Analizador LR.**

Estos analizadores sirven para algunas gramáticas libres de contexto, estos construyen un árbol de análisis sintáctico de las hojas a la raíz.

Este analizador utiliza la técnica de análisis por desplazamiento reducción.

Un analizador LR consta de lo siguiente:

- Una entrada
- Una salida
- Una tabla de análisis sintáctico, compuesta por dos partes (**ACCION Y GOTO**)

Se puede decir que un analizador LR transfiere los símbolos de su entrada a la pila, hasta que los símbolos que se encuentran en la parte superior sean iguales al lado derecho de alguna regla de reescritura de la gramática en la que se basa el analizador. En este punto el analizador podrá reemplazar los símbolos con el no terminal que se encuentra en el lado izquierdo de la regla de reescritura antes de transferir otros símbolos a la pila.

De esta manera la pila puede acumular cadenas de terminales y no terminales, que al mismo tiempo pueden ser reemplazados por no terminales que se encuentren en un “nivel más alto” de la gramática.

Para concluir, el contenido de la pila puede ser reducido al símbolo inicial de la gramática, indicando que los símbolos que han sido leídos hasta ese punto forman una cadena que puede derivarse con la gramática.

Al momento de desarrollar este método, podemos encontrarnos con dos problemas. El primero es referente al no determinismo, con los analizadores LR(k) si se presenta una opción, no se sabe si va a desplazar o reducir. Además, si la opción es reducir, puede haber más de una reducción.

Este problema podemos resolverlo a través de un pre análisis.

El segundo problema al que nos enfrentamos es por la interrogación de la pila, ya que para el pre análisis solo se dispone de un elemento de la pila, que es el elemento que se encuentra en la cima. La solución para este caso es atreves de la tabla de análisis sintáctico.

La tabla de un analizador sintáctico LR(k) se basa en la existencia de un autómata finito que puede aceptar exactamente las cadenas de símbolos de la gramática correspondiente que conducen a las operaciones de reducción.

Para obtener los resultados deseados es recomendable establecer una secuencia bien definida de los eventos guiados por los símbolos que se detectan en la cadena analizada.

El objetivo es comenzar con un proceso de análisis sintáctico, seguido de la ruta determinada por la cadena de entrada hasta encontrar con un estado de aceptación. La ruta es recorrida por el autómata finito, corresponde al patrón de símbolos que el analizador ha desplazado a la pila.

#### A su vez los LR se clasifican en:

- **SLR sencillo:** Es el más sencillo de implementar.
- **LR (1):** Es el más costoso de implementar, ya que incorpora el cálculo de los símbolos de lookhead a cada estado del AFD. A su vez es el que reconoce más gramáticas.
- **LALR:** Trata de conseguir aquellos beneficios del método LR(1) pero con el coste del método SLR.

### 3. PROPONER ALGORITMOS Y ESTRUCTURAS DE DATOS NECESARIAS PARA LA IMPLEMENTACIÓN DE UN PROTOTIPO DE ANALIZADOR SINTÁCTICO.

Las estructuras de datos que se proponen para la implementación del analizador sintáctico son las siguientes:

#### Árboles.

Son estructuras de datos ramificadas, es decir, no son lineales, estas pueden representarse como un conjunto de nodos que están entrelazados por medio de ramas.

El nodo base debe ser único, se le conoce como raíz y se establece en la parte superior. En esta estructura se representa una relación jerárquica a partir del nodo raíz en sentido vertical descendente, de esta manera se definen los diferentes niveles.

A partir de la raíz podemos llegar a cualquier nodo, esto se hace mediante el recorrido por las ramas y pasando por los diferentes niveles, de esta manera se establece un camino.

La relación que existe entre los nodos que se encuentran separados de forma inmediata, es decir por una rama, se denomina padre/hijo.

Cada nodo en el árbol representa una operación. Para esto, la operación más externa (la última que se realiza en la evaluación de una expresión) se encuentra en la raíz y los valores se almacenan en el nodo hoja.

Ejemplo.

Tenemos la expresión  $(5*10) + (20/2)$ , en este caso nuestra operación más externa es **+** que consecuentemente agrega  $(5*10)$  y  $(20/2)$ . A continuación, se muestra el árbol para esta expresión.

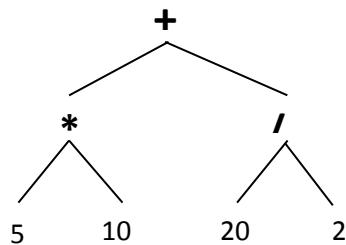


Imagen 3 Árbol sintáctico de la expresión  $(5*10) + (20/2)$

### Matriz bidireccional.

Para la tabla de análisis sintáctico LL(k) se propone utilizar una matriz bidimensional.

Las filas se etiquetan con los no terminales y las columnas con los terminales de la gramática.

Enseguida se añade la columna adicional FDC (fin de cadena). Los elementos **(m,n)** de la tabla indican la acción que debe de seguir cuando el no terminal **m** aparece en la cima de la pila y el símbolo de pre análisis es **n**.

Las tablas de análisis sintáctico simplifican la escritura del programa que efectúa el análisis y esto permite normalizar los algoritmos.

### Tabla de símbolos.

Las tablas de símbolos son estructuras de datos que almacenan toda la información de los identificadores de un lenguaje fuente. También se puede definir como una estructura de datos organizada en función de los identificadores que aparecen en el código fuente.

La tabla de símbolos que presentamos está conformada por las siguientes columnas:

**ID:** Este identifica de manera única a un elemento que se reconoce por el analizador. Además, es la localidad de memoria donde se encuentra el identificador. Por ejemplo: Para las variables tomamos el valor base de 140, posteriormente cuando se encuentra otra variable se simula la asignación a una localidad de memoria libre para así diferenciar de forma única a cada elemento.

**Etiqueta:** Representa el número del lexema.

**Lexema:** Hace referencia al token que se ha reconocido en el analizador.

**Descripción:** Proporciona un comentario acerca del elemento que se encuentra en la tabla.

A continuación, se muestra un ejemplo breve:

Etiqueta	ID	Lexema	Descripción
101		Init	Palabra reservada
117		(	delimitador_abre
118		)	delimitador_cierre
115		{	delimitador_abre
103		Int	Palabra reservada
140	141	\$suma	Identificador
110		;	delimitador
105		print	palabra reservada
117		(	delimitador_abre
140	142	\$suma	Identificador
118		)	delimitador_cierre
110		;	delimitador
116		}	delimitador_cierre

### Pseudocódigo.

1. Para almacenar un árbol en una estructura de datos, se tendrá que convertir en una jerarquía de objetos. Para esto se crea una interfaz base para los nodos del árbol y una clase para implementar los nodos.
2. La interfaz define un par de constantes de enteros estáticos que se pueden usar para identificar los diferentes nodos. Generar un método `getType()` para devolver el tipo de nodo. Después generar un método `getValue()` para devolver el resultado del subárbol que se genera con el nodo actual.
3. Para esto habrá dos constructores, uno tomará un valor doble de una constante y el otro un valor que se convierte en valor doble.
4. Se genera una clase que contiene las variables o nombre de las constantes.
5. La clase tiene un parámetro interno que almacena el nombre de la variable junto con su valor. El método `getValue()` devolverá el valor o generara una excepción.
6. Enseguida se crean dos constructores, uno se encarga de crear una secuencia vacía y el otro inmediatamente agrega el primer término a la secuencia.

### Algoritmo de análisis sintáctico LR

- Para las entradas:
  - Recibir una cadena de entrada w
  - La tabla de análisis sintáctico LR tendrá las funciones “acción” e “ir\_a” para la gramática G
- Para las salidas:
  - Si w está en L(G) es una derivación ascendente
  - Si no es así, indica un error.
- El método:
  - De manera inicial:
    - La pila contiene un estado inicial ( $S_0$ )
    - El buffer de entrada contiene w (la cadena de entrada seguida de la marca que indica el fin de la cadena).
  - En la ejecución del programa de análisis se realiza hasta encontrar una acción de aceptación o de error.

**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 CORRELACIONADOS CON LOS CASOS DE ESTUDIO CORRESPONDIENTES.**

## **PROGRAMA 1**

```
#te dice si dos números son amigos es decir el 220 sus divisores son 1  
,2,4,5,10,11,22,110,20,55 su suma es 284 y el numero 284 sus divisores son  
1,2,4,71,142 su suma es 220 y también nos dice si un número es perfecto o no es  
decir 6 sus divisores 1,2,3 su suma es 6 y 6 es igual al número entonces si es  
perfecto  
init() {  
    int $A=0,$B=0,$i=0,$sumaA=0,$sumaB=0;#declaramos las variables que se  
utilizan para los numeros amigos  
    print("NUMEROS AMIGOS");  
    print ("Teclea el primer numero: ");  
    int $A= read();  
    print("Teclea el segundo numero: ");  
    int $B= read();  
    while($i<=$A)  
    {  
        if($A%$i==0){$sumaA+=$i;}  
        $i++;  
    }  
    while($i<=$B)  
    {  
        if($B%$i==0){$sumaB+=$i;}  
        $i++ #no hemos puesto ;() al final de la instrucción  
    }  
    if($A==$sumaB && $B==$sumaA ) {  
        print("Los numeros".$A." Y ".$B." son amigos");  
    }  
    else  
    {  
        print("Los numeros ".$A." Y ".$B." son amigos");  
    }  
    print("NUMEROS PERFECTOS");  
    print("Teclea el numero a confirmar si es perfecto o no lo es: ");  
    $A=read();#leemos el numero desde teclado  
    $i=1;  
  
    while() {  
        if($A%i==0){$sumaA+=$i;}  
        i++;  
    }  
    if($A==$sumaA)  
    {  
        print("El numero ".$A." es un numero perfecto");  
    }  
    else  
    {  
        print("El numero ".$A." no es un numero perfecto");  
    }
```

```
}
```

```
}
```

## PROGRAMA 2

```
#nos da una serie de numero fibonaci
init() {
    int $n=0,$x=0,$y=1,$suma=0;#declaramos las variables que vamos a
necesar de tipo de dato entero
    print("Teclea el numero limite de la serie que deseas imprimir");
n = read(); #leemos desde teclado un valor entero y se asigna a una variable
entera
    if($n>0) {
        print($x);
        print($y);
        while($suma<=$n)
        {
            $suma=$x+$y;
            $x=$y;
            $y=$suma; if($suma<=$n)
            {
                print($suma);#imprimimos la serie fibonaci
            }
        }
    }
    int $z=@; #variable contiene caracteres inválidos.
    print("serie fibonacci");
    print("error de ortografía");
    while($z>=$z) #Error llaves no equilibradas
        print($suma);
    }
}
```

## PROGRAMA 3

```
# programa que hace el factorial de un numero
init() {
    int $factorial = 1,$num; #Declaramos las variables
    print("Ingrese un numero positivo para obtener su factorial");#Pedimos
un numero para calcular su factorial
    $num = read();#leemos desde teclado un valor entero y se asigna a una
variable entera
    if ($num==0) {
        print("se calculo 0!= " + $factorial); #si el numero es cero su
factorial es 1
    }
    else {
        if ($num < 0) { #Avisamos que solo se puede tener el factorial
de un numero que sea positivo
            print("Unicamente puede introducir numeros positivos");
        }
        else {
            while ($num > 0 #Error de parentesis {

$factorial=$factorial * $num;

$num--;
        }
    }
}
```

```

        }
        print(num + " != " + $factorial);#se muestra el factorial
    }
}

init(){
    print("Este programa de prueba no calcula el factorial de un numero");
    print("Hola mundo");
}

```

## PROGRAMA 4

```

#este programa da el puntaje apartir de cierta letra escogida por el usuario
init(){
print("Seleccione la calificacion(letra) para conocer su
puntaje\nA\nB\nC\nD\nE\nF");
String $letra=read();
if($letra=="A" || $letra=="a") { #se compara si la letra es mayúscula o
minúscula
    print("La calificacion es 100");
}
if($letra=="B" || $letra=="b") { #se compara si la letra es mayúscula o
minúscula

    print("La calificación es 90");
}
if($letra=="C" || $letra=="c") { #se compara si la letra es mayúscula o
minúscula

    print("La calificación es 80");
}
if($letra=="D" || $letra=="d") { #se compara si la letra es mayúscula o
minúscula

    print("La calificación es 75");
}
if($letra=="E" || $letra=="e") { #se compara si la letra es mayúscula o
minúscula

    print("La calificación es 70");
}
int $bandera=1;
if($bandera==1){
    fi($letra== "f")#la palabra reservada está mal escrita(if)
    {
        print("La calificación no es aprobatoria esta reprobado");
    }
}

```

## PROGRAMA 5

```

#programa que dice en que rango esta un numero del 1 al 100
init()#Error ausencia de llave

```

```

print("Dame un numero");
    int $i=read(); #lee una variable entera
    while($i<=100) {
        print(i);

            if($i==1){ #compara si la variable es un uno entonces es el
primer numero
                print("es el primer numero");
            }
            if($i>1 && $i<50){ #compara si la variable esta entre 1 y 50
                print("el numero es menor que 50");
                if($i>10 && $i<30){ #compara si la variable esta entre 10
y 30
                    print("el numero esta entre 10 y 30");
                }
            }
            if($i>50){ #compara si la variable es mayor a 50
                print("el numero es mayor a 50");
                if($i>50 && $i<70){ #compara si la variable esta entre 50 y
70
                    print("el numero esta entre 50 y 70");
                }
            }
        }
    }
}

```

## PROGRAMA 6

```

# Programa que lea un número entero y muestre si el número es múltiplo de 10
init { #Error ausencia de parentesis
    int $NN=0;
    int $N;
    print("Número entero: ");
    $N = read();
    if(N%10==0){
        print("Es múltiplo de 10");
    }
    else
        if($N==0){
            print("No es múltiplo de 10");
        }
    }
    while($NN>=0){
        if($NN%2==0){
            print("numero par");
        }
        else{
            print("numero impar");
        }
    }
    while($N>=0){ #compara si la variable NN es mayor o igual a cero y si N es
par o impar según sea el caso lo informa
        if($N%2==0){
            print("numero par");
        }
        else{

```

```
        print("numero impar");
    }
}
```

A continuación podemos ver como al introducir uno de los programas de prueba del lenguaje prototipo nos muestra la tabla de símbolos con la información necesaria como lo es una etiqueta el lexema y la descripción y en la segunda imagen podemos ver que si se eliminan por ejemplo los paréntesis el analizador sintáctico reconoce que existe un error informando la línea y la posición en la que se esperaba un paréntesis.

The screenshot shows a code editor interface with tabs for Run, Lexico, Sintactico, Semantico, and Guardar. The main area displays a Python script with syntax highlighting. The script performs various operations like reading integers from the user, checking if they are multiples of 10 or 2, and printing them as even or odd. The right side of the interface contains a table of tokens extracted from the script, with columns for Etiqueta (Label), ID, Lexema (Lexeme), and Descripcion (Description). The table includes entries for identifiers like \$NN, \$N, and 1001, as well as reserved words like if, print, and while.

Etiqueta	ID	Lexema	Descripcion
101		init	Palabra reservada
117		(	delimitador_abre
118		)	delimitador_cierre
115		{	delimitador_abre
103		int	Palabra reservada
140	141	\$NN	identificador
122		=	igual
1000	1001	0	Numero entero
110		:	delimitador
103		int	Palabra reservada
140	142	\$N	identificador
110		:	delimitador
105		print	Palabra reservada
117		(	delimitador_abre
2000	2001	"Número entero:"	Cadena de texto
118		)	delimitador_cierre
110		:	delimitador
116		}	delimitador_cierre
140	143	\$N	identificador
122		=	igual
104		read	Palabra reservada
117		(	delimitador_abre
118		)	delimitador_cierre
110		:	delimitador
110		:	delimitador
107		if	Palabra reservada

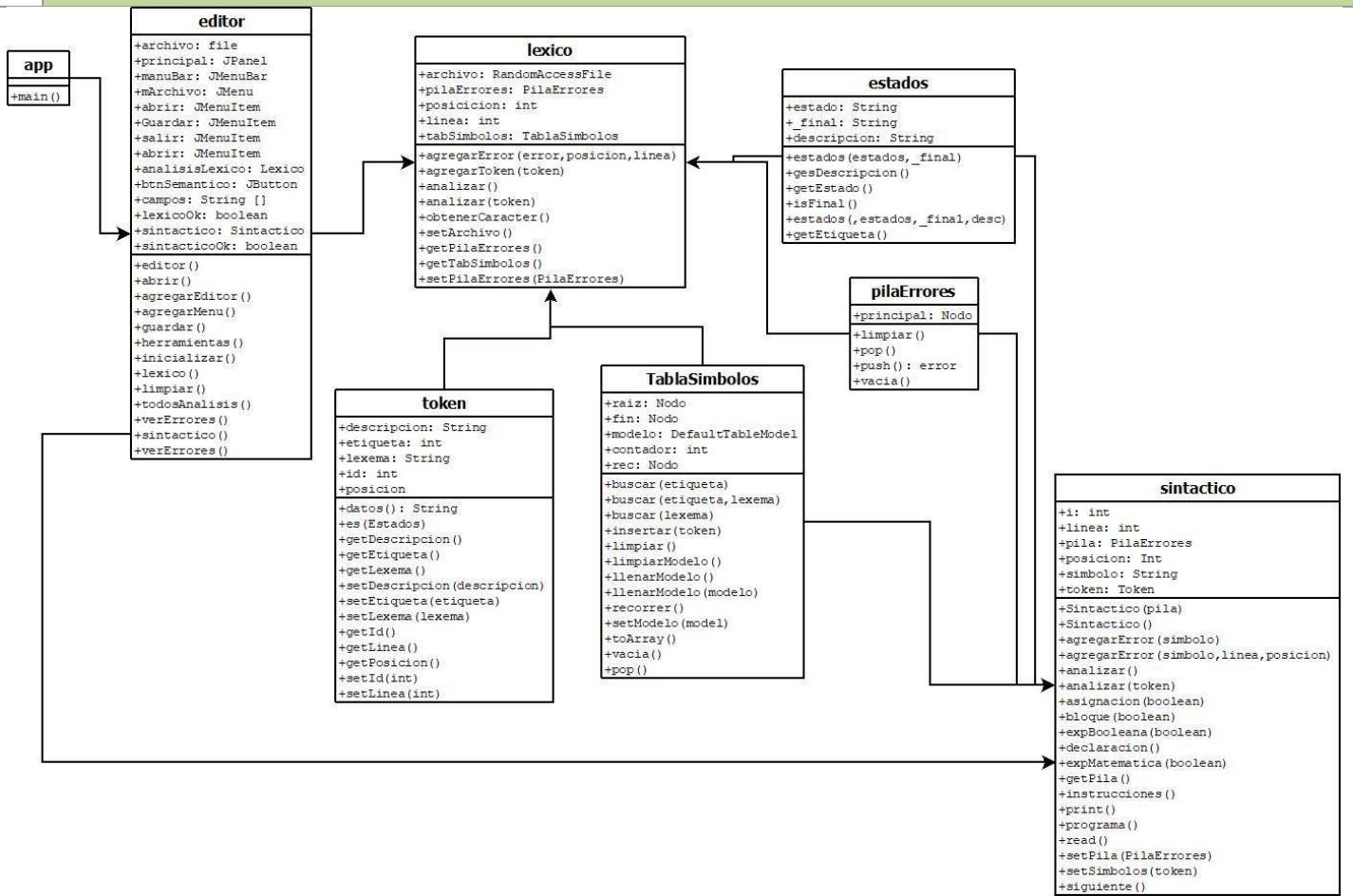
The screenshot shows a Java-based IDE interface. On the left, a script editor displays a Python-like script with syntax highlighting. The code checks if a number is a multiple of 10 and prints 'Es múltiplo de 10' or 'No es múltiplo de 10'. It also prints 'numero par' or 'numero impar' based on the value of \$N. A while loop at the end compares \$NN and \$N2. On the right, a table viewer displays tokens from the script, mapping them to labels, IDs, lexemes, and descriptions.

	Etiqueta	ID	Lexema	Descripción
1	init	101	init	Palabra reservada
2	{	115	{	delimitador_abre
3	int	103	int	Palabra reservada
4	\$NN;	140	\$NN	identificador
5	print("Número entero: ");	141	=	igual
6		1000	0	Número entero
7		110	:	delimitador
8	\$N = read();	103	int	Palabra reservada
9	if(\$N10==0){	140	\$N	identificador
10	print("Es múltiplo de 10");	142	:	delimitador
11	}	110	:	delimitador
12	else	103	int	Palabra reservada
13	if(\$N==0){	140	(	delimitador_abre
14	print("No es múltiplo de 10");	105	print	Palabra reservada
15	}	117	)	delimitador_cierra
16	while(\$NN>=0){	2000	"Número entero:"	Cadena de texto
17	if(\$N2==0){	118	)	delimitador_cierra
18	print("numero par");	110	:	delimitador
19	}	140	\$N	identificador
20	else{	143	=	igual
21	print("numero impar");	104	read	Palabra reservada
22	}	117	(	delimitador_abre
23	}	118	)	delimitador_cierra
24	while(\$N>=0){ #compara si la variable NN es mayor o igual a cero y si N es par o impar según sea el caso lo	110	:	delimitador
25	if(\$N2==0){	107	if	Palabra reservada
		117	(	delimitador_abre
		140	\$N	identificador
		1000	1002	10
				Número entero

Consola  
Se espera (Línea 2 posición 2)

4

## DIAGRAMA DE CLASES



5

## CALENDARIO DE ACTIVIDADES E INCIDENCIAS

### Calendario de Actividades.

#### Corrección del Análisis Léxico.

marzo de 2018						
do.	lu.	ma.	mi.	ju.	vi.	sá.
25	26	27	28	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Fecha	Actividad Planeada	Actividad Realizada	Tiempo Planeado	Tiempo Real
28-Marzo-18	Corrección del diagrama de clases	Corrección del diagrama de clases	1 hora	1 hora
28-Marzo-18	Corrección de los autómatas	Corrección de los autómatas	3 horas	3:30 horas
29-Marzo-18	Corrección de la pila de errores	Corrección de la pila de errores	2 horas	2 horas
30-Marzo-18	Corrección de la tabla de símbolos	Corrección de la tabla de símbolos	2 horas	1:30 horas
30-Marzo-18	Pruebas	Pruebas	1 hora	1 hora
31-Marzo-18	Pruebas	Pruebas	1 hora	1 hora

Se realizaron las correcciones indicadas para tener un analizador léxico eficiente, libre de errores y que cumpla con todas las observaciones realizadas.

#### Etapa de Investigación acerca del Análisis Sintáctico.

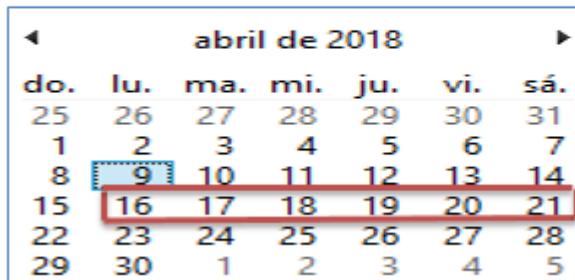
abril de 2018						
do.	lu.	ma.	mi.	ju.	vi.	sá.
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Para esta actividad se realizó una lectura previa de diversos materiales como: libros, documentos electrónicos. A continuación, se muestra la distribución de actividades.

Fecha	Actividad Planeada	Actividad Realizada	Tiempo Planeado	Tiempo Real
9-Abril-18	Lectura acerca del analizador sintáctico	Lectura acerca del analizador sintáctico	2 horas	2 horas
10-Abril-18	Revisión del Análisis Léxico	Revisión del Análisis Léxico	30 min	30 min
11-Abril-18	Investigación del punto 2 A) y B)	Investigación del punto 2 A) y B)	3 horas	4 horas
12-Abril-18	Investigación del punto 2 C) y D)	Investigación del punto 2 C) y D)	2 horas	4 horas
13-Abril-18	Investigación del punto 2 E)	Investigación del punto 2 E)	2 horas	2 horas

<b>13- Abril-18</b>	Proponer las estructuras de datos y realizar la tabla de símbolos perfectamente estructurada	Proponer las estructuras de datos y realizar la tabla de símbolos perfectamente estructurada	1 hora	1 hora
<b>13- Abril-18</b>	Generar programas de mínimo 25 líneas	Generar programas de mínimo 25 líneas	1 hora	1 hora
<b>14- Abril-18</b>	Generación de un catálogo de errores	Generación de un catálogo de errores	1 hora	1 hora
<b>14- Abril-18</b>	Diseño del diagrama de clases	Diseño del diagrama de clases	1 hora	1 hora

### **Etapa de programación del analizador sintáctico.**



Durante la semana indicada se realizó la programación de nuestro prototipo, ya teniendo un analizador léxico que funciona de acuerdo a las observaciones se procede a continuar con la siguiente etapa: Analizador Sintáctico.

A continuación, se muestra la bitácora de incidencias, es decir aquellos problemas que surgieron durante y la solución que se generó para concluir esta etapa.

#### **Bitácora de Incidencias.**

Fecha de error	Fecha de solución	Problema	Solución
<b>16-Abril-18</b>	16-Abril-18	La estructura de datos para realizar el análisis no funcionó adecuadamente	Cambiamos la estructura a un arreglo de tokens, lo cual nos permitirá ir de un lado a otro dentro de todos los tokens, además de que esto no permitirá no agregar más, dado que después del análisis léxico no se debe poder meter más tokens.
<b>17-Abril-18</b>	18-Abril-18	El análisis final no funciona de la mejor manera	Debugear cada análisis sintáctico, dado que se dividieron en análisis en analizar expresiones booleanas, estructuras de control, declaraciones, asignaciones y

			expresiones matemáticas.
<b>19-abril-18</b>	19-Abril-18	La inclusión de diferentes análisis falla	Pruebas unitarias de cada sub-análisis, y volver a programar las partes inservibles, para ver que parte no concuerda con todos los sub-análisis.
<b>20-Abril-18</b>	20-Abril-18	Falta de campos para la tabla de símbolos para el futuro análisis semántico	Agregar los campos faltantes, así como los errores posibles que se pueden dar durante el análisis sintáctico.

**6**

## DESARROLLO

Código fuente

The screenshot shows a Java development environment with two code editors open. The top editor, titled 'App.java', contains the following code:

```
1 package app;
2 public class App
3 {
4     public static void main( String[] args )
5     {
6         new Editor();
7     }
8 }
```

The bottom editor, titled 'Editor.java', contains the following code:

```
1 package app;
2
3 import java.awt.BorderLayout;
4 import java.awt.Color;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import java.io.File;
8 import java.io.FileNotFoundException;
9 import java.io.FileWriter;
10 import java.io.IOException;
11 import java.util.Scanner;
12
13 import javax.swing.JButton;
14 import javax.swing.JFileChooser;
15 import javax.swing.JFrame;
16 import javax.swing.JLabel;
17 import javax.swing.JMenu;
18 import javax.swing.JMenuBar;
19 import javax.swing.JMenuItem;
20 import javax.swing.JOptionPane;
21 import javax.swing.JPanel;
22 import javax.swing.JScrollPane;
23 import javax.swing.JTable;
24 import javax.swing.JTextArea;
25 import javax.swing.JToolBar;
26 import javax.swing.table.DefaultTableModel;
27
28 import org.fife.ui.rsyntaxtextarea.RSyntaxTextArea;
29 import org.fife.ui.rsyntaxtextarea.SyntaxConstants;
30 import org.fife.ui.rtextarea.RTextScrollPane;
```

Editor.java X

Source History | | | | |

```
32 import lexico.Lexico;
33 import sintactico.Sintactico;
34 import utils.PilaErrores;
35 import utils.TablaSimbolos;
36 import utils.Token;
37
38 public class Editor extends JFrame {
39     private static final long serialVersionUID = 1L;
40     private boolean lexicoOk = false, sintacticoOk = false;
41     private JButton btnLexico, btnSintactico, btnSemantic, btnRun, btnGuardar;
42
43     private Lexico analisisLexico;
44     private TablaSimbolos tablaSimbolos;
45     private File archivo;
46     private PilaErrores pilaErrores;
47     private RSyntaxTextArea editor;
48     private JPanel principal;
49     private JMenuBar menurBar;
50     private JMenu mArchivo;
51     private JMenuItem abrir, guardar, salir,nuevo;
52
53     private DefaultTableModel modelo;
54     private JTextArea consola;
55     private final String campos[] = { "Etiqueta","ID", "Lexema", "Descripcion" };
56
57     private Sintactico sintactico;
58
59     public Editor() {
60         pilaErrores = new PilaErrores();
61         tablaSimbolos = new TablaSimbolos();
62         analisisLexico = new Lexico(pilaErrores, tablaSimbolos);
```



Editor.java

Source History

```
90     });
91     guardar = new JMenuItem("Guardar");
92     guardar.addActionListener(new ActionListener() {
93
94         @Override
95         public void actionPerformed(ActionEvent e) {
96             guardar();
97         }
98     });
99     salir = new JMenuItem("Salir");
100    salir.addActionListener(new ActionListener() {
101
102        @Override
103        public void actionPerformed(ActionEvent arg0) {
104            System.exit(0);
105        }
106    });
107    nuevo = new JMenuItem("Nuevo");
108    nuevo.addActionListener(new ActionListener() {
109
110        @Override
111        public void actionPerformed(ActionEvent e) {
112            guardar();
113            archivo = null;
114            editor.setText("");
115        }
116    });
117    mArchivo.add(nuevo);
118    mArchivo.add(abrir);
119    mArchivo.add(guardar);
120
121    mArchivo.addSeparator();
```

The screenshot shows a Java code editor window titled "Editor.java". The code is written in Java and defines a class with methods for setting up a menu bar and adding an editor component. The code uses RTextScrollPane and JTable components.

```
121     mArchivo.addSeparator();
122     mArchivo.add(salir);
123     menurBar.add(mArchivo);
124
125     setJMenuBar(menurBar);
126 }
127
128 private void agregarEditor() {
129
130     editor = new RSyntaxTextArea(20, 60);
131     editor.setSyntaxEditingStyle(SyntaxConstants.SYNTAX_STYLE_JAVA);
132     editor.setCodeFoldingEnabled(true);
133     RTextScrollPane sp = new RTextScrollPane(editor);
134     principal.add(sp, BorderLayout.CENTER);
135
136     JTable tabla = new JTable();
137     tabla.setCellSelectionEnabled(false);
138     modelo = (DefaultTableModel) tabla.getModel();
139     for (String campo : campos)
140         modelo.addColumn(campo);
141     principal.add(new JScrollPane(tabla), BorderLayout.EAST);
142
143     consola = new JTextArea(5, 5);
144     consola.setEditable(false);
145     JLabel lblConsola = new JLabel("Consola");
146     JPanel pnlConsola = new JPanel(new BorderLayout());
147     pnlConsola.add(lblConsola, BorderLayout.NORTH);
148     pnlConsola.add(new JScrollPane(consola), BorderLayout.CENTER);
149     principal.add(pnlConsola, BorderLayout.SOUTH);
150 }
151 }
```

The screenshot shows a Java code editor window with the following details:

- Title Bar:** The title bar displays "Editor.java X".
- Toolbar:** A toolbar at the top contains icons for file operations (New, Open, Save, Print, Find, Replace, Cut, Copy, Paste, Undo, Redo), selection tools (Select All, Select Line, Select Block), and other common functions.
- Code Area:** The main area contains the following Java code:

```
153     private void herramientas() {
154         JToolBar herramientas = new JToolBar();
155         btnLexico = new JButton("Lexico");
156         btnLexico.addActionListener(new ActionListener() {
157             @Override
158             public void actionPerformed(ActionEvent e) {
159                 lexico();
160             }
161         });
162         btnSintactico = new JButton("Sintactico");
163         btnSintactico.setEnabled(false);
164         btnSintactico.addActionListener(new ActionListener() {
165             @Override
166             public void actionPerformed(ActionEvent e) {
167                 lexico();
168                 sintactico();
169             }
170         });
171         btnRun = new JButton("Run");
172         btnRun.addActionListener(new ActionListener() {
173             @Override
174             public void actionPerformed(ActionEvent e) {
175                 todosAnalisis();
176             }
177         });
178         btnSemantico = new JButton("Semantico");
179         btnSemantico.setEnabled(false);
180         btnGuardar = new JButton("Guardar");
181     }
```
- Annotations:** There are several annotations in the code, indicated by small yellow icons with numbers (e.g., 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181).



Editor.java

Source History

```
212     private void lexico() {
213         btnLexico.setBackground(Color.WHITE);
214         limpiar();
215         guardar();
216
217         try {
218             analisisLexico.setArchivo(archivo);
219             lexicoOk = analisisLexico.analizar();
220         } catch (Exception e) {
221             System.out.println(e);
222         }
223         if(lexicoOk)
224         {
225             btnSintactico.setEnabled(true);
226             btnLexico.setBackground(Color.GREEN);
227         }else {
228             verErrores();
229             btnSintactico.setEnabled(false);
230             btnLexico.setBackground(Color.red);
231         }
232         tablaSimbolos.llenarModelo(modelo);
233
234     }
235     private void sintactico()
236     {
237         if(lexicoOk)
238         {
239             Token[] simbolos = null;
240             simbolos = tablaSimbolos.toArray();
241             sintactico = null;
242             sintactico = new Sintactico();
```

Editor.java

Source History

```
235     private void sintactico()
236     {
237         if(lexicoOk)
238         {
239             Token[] simbolos = null;
240             simbolos = tablaSimbolos.toArray();
241             sintactico = null;
242             sintactico = new Sintactico();
243             sintacticoOk = sintactico.analizar(simbolos);
244             if(sintacticoOk)
245             {
246                 btnSintactico.setBackground(Color.GREEN);
247             }else
248             {
249                 btnSintactico.setBackground(Color.RED);
250                 pilaErrores = sintactico.getPila();
251                 verErrores();
252             }
253         }
254     }
255
256     private void verErrores()
257     {
258         String errores = "";
259         while(!pilaErrores.vacia())
260             errores += pilaErrores.pop()+"\n";
261         consola.setText(errores);
262     }
263
264     public void abrir() {
265         JFileChooser chooser = new JFileChooser();
```

The screenshot shows a Java code editor window with the file "Editor.java" open. The code implements a text editor with methods for opening and saving files.

```
263
264     public void abrir() {
265         JFileChooser chooser = new JFileChooser();
266         chooser.showOpenDialog(this);
267         archivo = chooser.getSelectedFile();
268         if (archivo != null) {
269             Scanner lector;
270             try {
271                 lector = new Scanner(archivo);
272                 String cadena = "";
273                 while (lector.hasNextLine()) {
274                     cadena += lector.nextLine() + "\n";
275                 }
276                 editor.setText(cadena);
277             } catch (FileNotFoundException e) {
278                 System.out.println(e.getMessage());
279             }
280         }
281     }
282
283     public void guardar() {
284         if (archivo == null) {
285             JFileChooser chooser = new JFileChooser();
286             chooser.showSaveDialog(this);
287             archivo = chooser.getSelectedFile();
288         }
289         FileWriter escritor;
290         if(archivo != null)
```

Editor.java

Source History

```
280
281
282
283    }
284    public void guardar() {
285        if (archivo == null) {
286            JFileChooser chooser = new JFileChooser();
287            chooser.showSaveDialog(this);
288            archivo = chooser.getSelectedFile();
289        }
290        FileWriter escritor;
291        if(archivo != null)
292        try {
293            escritor = new FileWriter(archivo);
294            escritor.write(editor.getText());
295            escritor.close();
296            // JOptionPane.showMessageDialog(null, "Guardado");
297        } catch (IOException e) {
298            JOptionPane.showMessageDialog(null, "No se ha podido guardar");
299            e.printStackTrace();
300        }
301    }
302
303}
304
```

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows the file name "Estados.java".
- Toolbar:** Includes standard icons for file operations like Open, Save, Print, and Search.
- Source Tab:** The active tab, indicated by a blue border.
- History Tab:** A secondary tab.
- Code Area:** Displays the following Java code:

```
1 package lexico;
2
3 public enum Estados
4 {
5     Q0("Q0", false),
6     Q1("Q1", false),
7     Q2("Q2", false),
8     Q3("Q3", false),
9     INT(103, "int", true, "Palabra reservada"),
10    IF(107, "if", true, "Palabra reservada"),
11    Q6("Q6", false),
12    INIT(101, "init", true, "Palabra reservada"),
13    Q8("Q8", false),
14    Q9("Q9", false),
15    Q10("Q10", false),
16    Q11("Q11", false),
17    WHILE(106, "while", true, "Palabra reservada"),
18    Q13("Q13", false),
19    Q14("Q14", false),
20    Q15("Q15", false),
21    Q16("Q16", false),
22    Q17("Q17", false),
23    Q18("Q18", false),
24    STRING(102, "String", true, "Palabra reservada"),
25    Q20("Q20", false),
26    Q21("Q21", false),
27    Q22("Q22", false),
28
29    Q23("Q23", false),
30    Q24("Q24", false),
31    Q25("Q25", false),
}
```

Estados.java X

Source History | | | | |

```
32     Q26("Q26",false),
33     PRINT(105,"print",true,"Palabra reservada"),
34
35     Q27("Q27",false),
36     Q28("Q28",false),
37     Q29("Q29",false),
38     READ(104,"read",true,"Palabra reservada"),
39
40
41     ELSE(108,"else",true,"Palabra reservada"),
42     DIVISION(133,"/",true,"Operador aritmetico"),
43     MULTIPLICACION(132,"*",true,"Operador aritmetico"),
44     RESTA(131,"-",true,"Operador aritmetico"),
45     SUMA(130,"+",true,"Operador aritmetico"),
46
47     IGUAL(122,"=",true,"igual"),
48     MENOR(120,"<",true,"menor que"),
49     MAYOR(121,">",true,"mayor que"),
50
51     AND(122,"&",true,"and"),
52     OR(123,"|",true,"or"),
53
54     ABRE_LLAVE(115,"{",true,"delimitador_abre"),
55     ABRE_PARENTESIS(117,"(",true,"delimitador_abre"),
56     CIERRA_LLAVE(116,"}",true,"delimitador_cierre"),
57     CIERRA_PARENTESIS(118,")",true,"delimitador_cierre"),
58     PUNTO_COMA(110,";",true,"delimitador"),
59
60     ID(140),
61     VALOR_INT(1000),
62     VALOR_STRING(2000)
```

Estados.java X

Source History | | |

```
63;
64;
65;
66    private final String estado;
67    private final boolean _final;
68    private final String descripcion;
69    private final int etiqueta;
70;
71    Estados (int etiqueta, String estado, boolean _final, String desc)
72    {
73        this.etiqueta = etiqueta;
74        this.estado = estado;
75        this._final = _final;
76        this.descripcion = desc;
77    }
78;
79    Estados (String estado, boolean _final, String desc)
80    {
81        etiqueta = 0;
82        this.estado = estado;
83        this._final = _final;
84        this.descripcion = desc;
85    }
86    Estados (int etiqueta)
87    {
88        this.etiqueta = etiqueta;
89        _final = false;
90        descripcion = "";
91        estado = "";
92    }
93    Estados (String estado, boolean _final){}
```

Estados.java

```
93     Estados (String estado,boolean _final){
94         etiqueta = 0;
95         this.estado = estado;
96         this.descripcion = "";
97         this._final = _final;
98     }
99
100    public String getEstado() {
101        return estado;
102    }
103    public boolean isFinal() {
104        return _final;
105    }
106
107    public String getDescripcion() {
108        return descripcion;
109    }
110
111    public int getEtiqueta()
112    {
113        return etiqueta;
114    }
115}
116
117
```

Lexico.java

```
1 package lexico;
2
3 import java.io.File;
4 import java.io.FileNotFoundException;
5 import java.io.IOException;
6 import java.io.RandomAccessFile;
7 import java.util.HashMap;
8 import java.util.LinkedHashMap;
9 import java.util.Map;
10
11 import utils.PilaErrores;
12 import utils.TablaSimbolos;
13 import utils.Token;
14
15 public class Lexico {
16     private RandomAccessFile archivo;
17     private PilaErrores pilaErrores;
18     private int posicion = 0, linea = 1, posicionToken = 0, ID = 0, VALOR_STRING = 0, VALOR_INT;
19
20     private TablaSimbolos tabSimbolos;
21     static private LinkedHashMap<Estados, Map<Character, Estados>> automata = new LinkedHashMap<Estados, Map<Character, Estados>>;
22     static {
23         Map<Character, Estados> transicion = new HashMap<Character, Estados>();
24         // De q0
25         transicion.put('i', Estados.Q1);
26         transicion.put('S', Estados.Q13);
27         transicion.put('w', Estados.Q8);
28         transicion.put('e', Estados.Q20);
29         transicion.put('p', Estados.Q23);
30         transicion.put('r', Estados.Q27);
```

Lexico.java

```

24    // De q0
25    transicion.put('i', Estados.Q1);
26    transicion.put('S', Estados.Q13);
27    transicion.put('w', Estados.Q8);
28    transicion.put('e', Estados.Q20);
29    transicion.put('p', Estados.Q23);
30    transicion.put('r', Estados.Q27);
31
32    transicion.put('+', Estados.SUMA);
33    transicion.put('-', Estados.RESTA);
34    transicion.put('*', Estados.MULTIPLICACION);
35    transicion.put('/', Estados.DIVISION);
36
37    transicion.put('>', Estados.MAYOR);
38    transicion.put('<', Estados.MENOR);
39    transicion.put('=', Estados.IGUAL);
40
41    transicion.put('&', Estados.AND);
42    transicion.put('|', Estados.OR);
43    transicion.put('{', Estados.ABRE_LLAVE);
44    transicion.put('}', Estados.CIERRA_LLAVE);
45    transicion.put('(', Estados.ABRE_PARENTESIS);
46    transicion.put(')', Estados.CIERRA_PARENTESIS);
47    transicion.put(';', Estados.PUNTO_COMA);
48
49
50
51
52
53    automata.put(Estados.Q0, transicion);
54    transicion = null;
55    // De q1
56    transicion = new HashMap<>();
57    transicion.put('n', Estados.Q2);

```

Lexico.java

```

54
55
56
57    transicion.put('f', Estados.IF);
58    automata.put(Estados.Q1, transicion);
59    transicion = null;
60    // De Q2
61    transicion = new HashMap<>();
62    transicion.put('t', Estados.Q3);
63    transicion.put('i', Estados.Q6);
64    automata.put(Estados.Q2, transicion);
65    transicion = null;
66    // De Q6
67    transicion = new HashMap<>();
68    transicion.put('t', Estados.INIT);
69    automata.put(Estados.Q6, transicion);
70    transicion = null;
71    // De Q3
72    transicion = new HashMap<>();
73    transicion.put(' ', Estados.INT);
74    automata.put(Estados.Q3, transicion);
75    transicion = null;
76    // De q13
77    transicion = new HashMap<>();
78    transicion.put('t', Estados.Q14);
79    automata.put(Estados.Q13, transicion);
80    transicion = null;
81    // De q14
82    transicion = new HashMap<>();
83    transicion.put('r', Estados.Q15);
84    automata.put(Estados.Q14, transicion);
85    transicion = null;

```

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Lexico.java
- Tab Bar:** Source (selected), History
- Toolbar:** Includes icons for cut, copy, paste, find, search, and other common file operations.
- Code Area:** Displays Java code for state transitions in a DFA.

```
83     // De Q15
84     transicion = new HashMap<>();
85     transicion.put('i', Estados.Q16);
86     automata.put(Estados.Q15, transicion);
87     transicion = null;
88     // De Q16
89     transicion = new HashMap<>();
90     transicion.put('n', Estados.Q17);
91     automata.put(Estados.Q16, transicion);
92     transicion = null;
93     // De Q17
94     transicion = new HashMap<>();
95     transicion.put('g', Estados.Q18);
96     automata.put(Estados.Q17, transicion);
97     transicion = null;
98     // De Q18
99     transicion = new HashMap<>();
100    transicion.put(' ', Estados.STRING);
101    automata.put(Estados.Q18, transicion);
102    transicion = null;
103    // De Q8
104    transicion = new HashMap<>();
105    transicion.put('h', Estados.Q9);
106    automata.put(Estados.Q8, transicion);
107    transicion = null;
108
109    // De Q9
110    transicion = new HashMap<>();
111    transicion.put('i', Estados.Q10);
112    automata.put(Estados.Q9, transicion);
```

The screenshot shows a Java code editor window titled "Lexico.java". The "Source" tab is selected. The code is a snippet of Java code related to state transitions in a DFA. It uses nested loops and state identifiers like Q10, Q11, Q20, and Q21. The code is annotated with several small yellow icons of lightbulbs and A/B test symbols, likely from a tool like SonarQube.

```
109
110         // De Q9
111         transicion = new HashMap<>();
112         transicion.put('i', Estados.Q10);
113         automata.put(Estados.Q9, transicion);
114         transicion = null;
115
116         // De Q10
117         transicion = new HashMap<>();
118         transicion.put('l', Estados.Q11);
119         automata.put(Estados.Q10, transicion);
120         transicion = null;
121
122         // De Q11
123         transicion = new HashMap<>();
124         transicion.put('e', Estados.WHILE);
125         automata.put(Estados.Q11, transicion);
126         transicion = null;
127
128         // De Q20
129         transicion = new HashMap<>();
130         transicion.put('l', Estados.Q21);
131         automata.put(Estados.Q20, transicion);
132         transicion = null;
133
134         // De Q21
135         transicion = new HashMap<>();
136         transicion.put('s', Estados.Q22);
137         automata.put(Estados.Q21, transicion);
138         transicion = null;
```

Lexico.java

Source History

```

133
134 // De Q21
135 transicion = new HashMap<>();
136 transicion.put('s', Estados.Q22);
137 automata.put(Estados.Q21, transicion);
138 transicion = null;
139
140 // De Q22
141 transicion = new HashMap<>();
142 transicion.put('e', Estados.ELSE);
143 automata.put(Estados.Q22, transicion);
144 transicion = null;
145 // De Q23
146 transicion = new HashMap<>();
147 transicion.put('r', Estados.Q24);
148 automata.put(Estados.Q23, transicion);
149 transacion = null;
150 // De Q24
151 transacion = new HashMap<>();
152 transacion.put('i', Estados.Q25);
153 automata.put(Estados.Q24, transacion);
154 transacion = null;
155 // De Q25
156 transacion = new HashMap<>();
157 transacion.put('n', Estados.Q26);
158 automata.put(Estados.Q25, transacion);
159 transacion = null;
160 // De Q26
161 transacion = new HashMap<>();
162 transacion.put('t', Estados.PRINT);
163 automata.put(Estados.Q26, transacion);

```

Lexico.java X

Source History

```
166 // De Q27
167 transicion = new HashMap<>();
168 transicion.put('e', Estados.Q28);
169 automata.put(Estados.Q27, transicion);
170 transicion = null;
171 // De Q28
172 transicion = new HashMap<>();
173 transicion.put('a', Estados.Q29);
174 automata.put(Estados.Q28, transicion);
175 transicion = null;
176
177 // De Q29
178 transicion = new HashMap<>();
179 transicion.put('d', Estados.READ);
180 automata.put(Estados.Q29, transicion);
181 transicion = null;
182 }
183
184 public Lexico() {
185 }
186
187 public Lexico(PilaErrores pilaErrores, TablaSimbolos tabSimbolos) {
188     this.archivo = null;
189     this.pilaErrores = pilaErrores;
190     this.tabSimbolos = tabSimbolos;
191 }
192
193 public void setArchivo(File archivo) throws FileNotFoundException {
194     if (archivo != null)
```

Lexico.java X

Source History

```
194     public void setArchivo(File archivo) throws FileNotFoundException {
195         if (archivo != null)
196             this.archivo = new RandomAccessFile(archivo, "r");
197     }
198
199     public Lexico(File archivo, PilaErrores pilaErrores) throws FileNotFoundException {
200         this.archivo = new RandomAccessFile(archivo, "r");
201         this.pilaErrores = pilaErrores;
202     }
203
204     public Lexico(File archivo, PilaErrores pilaErrores, TablaSimbolos tabSimbolos) throws FileNotFoundException {
205         this.archivo = new RandomAccessFile(archivo, "r");
206         this.pilaErrores = pilaErrores;
207         this.tabSimbolos = tabSimbolos;
208     }
209
210     public Lexico(File archivo) throws FileNotFoundException {
211         pilaErrores = new PilaErrores();
212         tabSimbolos = new TablaSimbolos();
213         this.archivo = new RandomAccessFile(archivo, "r");
214
215     }
216
217     public void agregarError(String error, int linea, int posicion) {
218         if (pilaErrores == null)
219             pilaErrores = new PilaErrores();
220         pilaErrores.push("Error " + error + " linea " + linea + " posicion " + posicion);
221     }
222 }
```

Lexico.java

```
222
223
224
225     public boolean analizar() throws IOException {
226         Estados actual;
227         actual = Estados.Q0;
228         Token token;
229         linea = 1;
230         posicion = 0;
231         posicionToken = 0;
232         VALOR_INT = 0;
233         VALOR_STRING = 0;
234         ID = 0;
235         Map<Character, Estados> transiciones;
236         StringBuilder sb = new StringBuilder();
237         char caracter;
238         do {
239             caracter = obtenerCaracter();
240
241             if (caracter == '#') {
242                 do {
243                     caracter = obtenerCaracter();
244                 } while (caracter != '\n');
245                 continue;
246             }
247
248             if (caracter == '$') {
249                 ID++;
250                 do {
251                     sb.append(caracter);
252                     caracter = obtenerCaracter();
253                 } while (caracter != '\n');
254                 agregarToken(new Token(Estados.ID.getEtiqueta(),(Estados.ID.getEtiqueta()+ID), sb.toString(), "identificador"));
255                 sb = new StringBuilder();
256             }
257
258             if (Character.isDigit(caracter)) {
259                 VALOR_INT++;
260                 do {
261                     sb.append(caracter);
262                     caracter = obtenerCaracter();
263                 } while (Character.isDigit(caracter));
264                 posicionToken++;
265                 agregarToken(new Token(Estados.VALOR_INT.getEtiqueta(),Estados.VALOR_INT.getEtiqueta() + VALOR_INT, sb.toString(), "entero"));
266                 sb = new StringBuilder();
267             }
268
269             if (caracter == '\"') {
270                 VALOR_STRING++;
271                 do {
272                     sb.append(caracter);
273                     caracter = obtenerCaracter();
274                 } while (caracter != '\"' && caracter != '\u0000');
275                 sb.append(caracter);
276                 if (caracter == '\"')
277                     linea++;
278             }
279         } while (caracter != '\n');
280         return true;
281     }
282 }
```

Lexico.java

```
248             if (caracter == '$') {
249                 ID++;
250                 do {
251                     sb.append(caracter);
252                     caracter = obtenerCaracter();
253                 } while (Character.isLetter(caracter));
254                 posicionToken++;
255                 agregarToken(new Token(Estados.ID.getEtiqueta(),(Estados.ID.getEtiqueta()+ID), sb.toString(), "identificador"));
256                 sb = new StringBuilder();
257             }
258
259             if (Character.isDigit(caracter)) {
260                 VALOR_INT++;
261                 do {
262                     sb.append(caracter);
263                     caracter = obtenerCaracter();
264                 } while (Character.isDigit(caracter));
265                 posicionToken++;
266                 agregarToken(new Token(Estados.VALOR_INT.getEtiqueta(),Estados.VALOR_INT.getEtiqueta() + VALOR_INT, sb.toString(), "entero"));
267                 sb = new StringBuilder();
268             }
269
270             if (caracter == '\"') {
271                 VALOR_STRING++;
272                 do {
273                     sb.append(caracter);
274                     caracter = obtenerCaracter();
275                 } while (caracter != '\"' && caracter != '\u0000');
276                 sb.append(caracter);
277                 if (caracter == '\"')
278                     linea++;
279             }
280         } while (caracter != '\n');
281         return true;
282     }
283 }
```

```
277     if (caracter == '') {
278         posicionToken++;
279         agregarToken(new Token(Estados.VALOR_STRING.getEtiqueta(), Estados.VALOR_STRING.getEtiqueta() +
280     } else {
281         caracter = '\u0000';
282         agregarError("Se esperaba \" ", linea, posicion);
283     }
284     sb = new StringBuilder();
285     continue;
286 }
287 transiciones = automata.get(actual);
288 actual = transiciones.get(caracter);
289 if (actual == null & Character.isWhitespace(caracter)) {
290     actual = Estados.Q0;
291     continue;
292 }
293
294 if (actual != null & actual.isFinal()) {
295     posicionToken++;
296     token = new Token(actual.getEtiqueta(), actual.getEstado(), actual.getDescripcion(), linea, posicionToken);
297     agregarToken(token);
298     token = null;
299     actual = Estados.Q0;
300     continue;
301 }
302 if (actual == null & caracter != '\u0000') {
303     agregarError("Caracter no valido " + caracter, linea, posicion);
304     actual = Estados.Q0;
305 }
306 } while (caracter != '\u0000');

295     posicionToken++;
296     token = new Token(actual.getEtiqueta(), actual.getEstado(), actual.getDescripcion(), linea, posicionToken);
297     agregarToken(token);
298     token = null;
299     actual = Estados.Q0;
300     continue;
301 }
302 if (actual == null & caracter != '\u0000') {
303     agregarError("Caracter no valido " + caracter, linea, posicion);
304     actual = Estados.Q0;
305 }
306 } while (caracter != '\u0000');
307 return (pilaErrores.vacia()) ? true : false;
308 }

310 public void agregarToken(Token token) {
311     if (tabSimbolos == null)
312         tabSimbolos = new TablaSimbolos();
313     tabSimbolos.insertar(token);
314 }
316
317 public boolean analizar(File archivo) throws IOException {
318     this.archivo = new RandomAccessFile(archivo, "r");
319     return analizar();
320 }
321
322 public char obtenerCaracter() throws IOException {
323     char caracter;
324     if (archivo != null & archivo.getFilePointer() < archivo.length()) {
325         caracter = (char) archivo.readByte();
```

The screenshot shows a Java code editor window with the file "Lexico.java" open. The code is a lexical analyzer implementation. The editor features syntax highlighting and code folding. The code itself is as follows:

```
324     if (archivo != null && archivo.getFilePointer() < archivo.length()) {
325         caracter = (char) archivo.readByte();
326         posicion++;
327         if (caracter == '\n') {
328             linea++;
329             posicion = 1;
330             posicionToken = 0;
331         }
332         return caracter;
333     } else if (archivo != null && archivo.getFilePointer() == archivo.length()) {
334         archivo.close();
335         archivo = null;
336     }
337     return '\u0000';
338 }
339
340 public PilaErrores getPilaErrores() {
341     return pilaErrores;
342 }
343
344 public void setPilaErrores(PilaErrores pilaErrores) {
345     this.pilaErrores = pilaErrores;
346 }
347
348 public TablaSimbolos getTabSimbolos() {
349     return tabSimbolos;
350 }
351
352 public void setTabSimbolos(TablaSimbolos tabSimbolos) {
353     this.tabSimbolos = tabSimbolos;
```

The screenshot shows a Java code editor window with the following details:

- Title Bar:** Shows the file name "Sintactico.java".
- Toolbar:** Includes icons for Source, History, and various navigation and search functions.
- Code Area:** Displays the Java code for the `Sintactico` class. The code includes imports for `lexico.Estados`, `utils.PilaErrores`, and `utils.Token`. It defines a constructor that initializes a stack and a token array, and two methods for analyzing programs and tokens respectively.

```
1 package sintactico;
2
3 import lexico.Estados;
4 import utils.PilaErrores;
5 import utils.Token;
6
7 public class Sintactico {
8     private PilaErrores pila;
9     private Token[] simbolos;
10    private Token token;
11    int i = 0;
12    private int linea = 0, posicion = 0;
13    public Sintactico(PilaErrores pila) {
14        this.pila = pila;
15    }
16
17    public Sintactico()
18    {
19        i = 0;;
20        pila = new PilaErrores();
21    }
22    public boolean analizar() {
23        i = 0;
24        programa();
25        return pila.vacia();
26    }
27    public boolean analizar(Token[] simbolos)
28    {
29        this.simbolos = simbolos;
30        return analizar();
31    }
}
```

```
33
34     public void programa() {
35         siguiente();
36         if (token != null && token.es(Estados.INIT)) {
37             siguiente();
38             if (token != null && token.es(Estados.ABRE_PARENTESIS)) {
39                 siguiente();
40                 if (token != null && token.es(Estados.CIERRA_PARENTESIS)) {
41                     bloque(true);
42                 } else
43                     agregarError(")");
44             } else
45                 agregarError("(");
46         } else
47             agregarError("INIT");
48     }
49
50     void bloque(boolean init) {
51         siguiente();
52         if (token != null && token.es(Estados.ABRE_LLAVE)) {
53             instrucciones();
54             siguiente();
55             if (token != null && token.es(Estados.CIERRA_LLAVE)) {
56                 if (!init)
57                     instrucciones();
58             } else
59                 agregarError("}");
60         } else
61             agregarError("{");
62     }
63 }
```

The screenshot shows a Java code editor interface with a tab labeled "Sintactico.java". The code is a part of a class named "Sintactico" and contains methods for parsing instructions, reading tokens, and printing results. The code uses nested loops and conditional statements to handle different types of tokens based on their state.

```
64     void instrucciones() {
65         asignacion(false);
66         declaracion();
67         _while();
68         _if();
69         print();
70     }
71 }
72
73 void read()
74 {
75     siguiente();
76     if(token != null && token.es(Estados.READ))
77     {
78         siguiente();
79         if(token != null && token.es(Estados.ABRE_PARENTESIS))
80         {
81             siguiente();
82             if(token != null && token.es(Estados.CIERRA_PARENTESIS))
83             {
84                 siguiente();
85             } else agregarError("(");
86             } else agregarError(")");
87             i--;
88     }
89 }
90 void print()
91 {
92     siguiente();
93     if(token != null && token.es(Estados.PRINT))
94     {
```



Sintactico.java

```
121     } else
122         agregarError(")");
123     } else
124         agregarError("(");
125     } else
126         i--;
127     }
128
129     void _if() {
130         siguiente();
131         if (token != null && token.es(Estados.IF)) {
132             siguiente();
133             if (token != null && token.es(Estados.ABRE_PARENTESIS)) {
134                 expBooleana(false);
135                 siguiente();
136                 if (token.es(Estados.CIERRA_PARENTESIS)) {
137                     bloque(false);
138                 } else
139                     agregarError(")");
140             } else
141                 agregarError("(");
142         } else
143             i--;
144     }
145
146
147     void declaracion()
148     {
149         siguiente();
150         if(token != null && (token.es(Estados.STRING) || token.es(Estados.INT)))
151         {
152             asignacion(true);
153         } else i--;
154     }
155
156     void asignacion(boolean declaracion)
157     {
158         siguiente();
159         if(token != null && token.es(Estados.ID))
160         {
161             siguiente();
162             if(token != null && token.es(Estados.IGUAL))
163             {
164                 expMatematica(false);
165             }
166             if(token != null && token.es(Estados.READ))
167             {
168                 i--;
169                 read();
170             }
171             if(token!= null && token.es(Estados.PUNTO_COMA))
172             {
173                 instrucciones();
174             } else agregarError(";");
175         } else if(declaracion) agregarError("ID"); else i--;
176     }
177
178
179     void expMatematica(boolean otra) {
180         siguiente();
181         if (token != null && (token.es(Estados.ID) || token.es(Estados.VALOR_INT) || token.es(Estados.VALOR_STRING))) {
182             siguiente();
183         }
184     }
185 }
```

Sintactico.java

```
152         asignacion(true);
153     } else i--;
154
155
156     void asignacion(boolean declaracion)
157     {
158         siguiente();
159         if(token != null && token.es(Estados.ID))
160         {
161             siguiente();
162             if(token != null && token.es(Estados.IGUAL))
163             {
164                 expMatematica(false);
165             }
166             if(token != null && token.es(Estados.READ))
167             {
168                 i--;
169                 read();
170             }
171             if(token!= null && token.es(Estados.PUNTO_COMA))
172             {
173                 instrucciones();
174             } else agregarError(";");
175         } else if(declaracion) agregarError("ID"); else i--;
176     }
177
178
179     void expMatematica(boolean otra) {
180         siguiente();
181         if (token != null && (token.es(Estados.ID) || token.es(Estados.VALOR_INT) || token.es(Estados.VALOR_STRING))) {
182             siguiente();
183         }
184     }
185 }
```

```
181     if (token != null && (token.es(Estados.ID) || token.es(Estados.VALOR_INT) || token.es(Estados.VALOR_STRING))) {
182         siguiente();
183         if (token != null && (token.es(Estados.SUMA) || token.es(Estados.RESTA) || token.es(Estados.DIVISION)
184             || token.es(Estados.MULTIPLICACION))) {
185             expMatematica(true);
186         }
187     } else if (otra) {
188         i--;
189         agregarError("INT | VALOR_INT");
190     }
191 }
192
193 void expBooleana(boolean otra) {
194     siguiente();
195     if (token != null && (token.es(Estados.INT) || token.es(Estados.ID) || token.es(Estados.VALOR_INT))) {
196         siguiente();
197         if (token != null && (token.es(Estados.MENOR) || token.es(Estados.MAYOR) || token.es(Estados.IGUAL))) {
198             siguiente();
199             if (token != null && (token.es(Estados.INT) || token.es(Estados.ID) || token.es(Estados.VALOR_INT))) {
200                 siguiente();
201                 if (token != null && (token.es(Estados.AND) || token.es(Estados.OR))) {
202                     expBooleana(true);
203                 } else {
204                     i--;
205                     return;
206                 }
207             } else
208                 agregarError("INT | ID");
209         } else
210             agregarError("INT | VALOR_INT");
211
212     } else if (otra)
213         agregarError("INT | ID");
214 }
215
216 public void agregarError(String simbolo) {
217     agregarError(simbolo, linea, posicion);
218 }
219
220 public void agregarError(String simbolo, int linea, int posicion) {
221     if (pila == null)
222         pila = new PilaErrores();
223     pila.push("Se esperaba " + simbolo + " Linea " + linea + " posicion " + posicion);
224 }
225
226 private void siguiente() {
227
228     if (i < simbolos.length) {
229         token = simbolos[i];
230         linea = token.getLinea();
231         posicion = token.getPosicion();
232         //System.out.println(token.getLexema());
233     } else
234         token = null;
235     i++;
236 }
237
238 public PilaErrores getPila() {
239     return pila;
```

The screenshot shows a Java code editor window with the following details:

- Title Bar:** "Sintactico.java X"
- Menu Bar:** "Source" and "History" are visible.
- Toolbar:** Includes icons for copy, paste, cut, find, search, and other standard file operations.
- Code Area:** Displays the following Java code (approximate line numbers 232-251):

```
232         //System.out.println(token.getLexema());
233     } else
234     token = null;
235     i++;
236 }
237
238 public PilaErrores getPila() {
239     return pila;
240 }
241
242 public void setSimbolos(Token[] simbolos) {
243     this.simbolos = simbolos;
244 }
245
246 public void setPila(PilaErrores pila) {
247     this.pila = pila;
248 }
249
250 }
251
```

The screenshot shows a Java code editor window with the following details:

- Title Bar:** The title bar displays "PilaErrores.java" and includes standard window controls.
- Toolbar:** A toolbar at the top contains various icons for file operations like new, open, save, and search.
- Source Tab:** The "Source" tab is selected, indicating the current view of the code.
- Code Area:** The main area contains the following Java code:

```
1 package utils;
2
3 public class PilaErrores {
4
5     private Nodo principal;
6     public PilaErrores()
7     {
8         principal = null;
9     }
10
11    public static void main(String... arg)
12    {
13        PilaErrores pila = new PilaErrores();
14        System.out.println(pila.vacia());
15        pila.push("Hola");
16        pila.push("Holai");
17        pila.push("Hola2");
18        pila.push("Hola3");
19        while(!pila.vacia())
20        {
21            System.out.println(pila.pop());
22        }
23
24    }
25
26
27    public String pop() {
28        String error = "";
29        if(!vacia())
30        {
31            error = principal.error;
```

The screenshot shows a Java code editor window with the following code:

```
30
31     error = principal.error;
32     principal = principal.sig;
33 }
34 return error;
35 }

38 public void push(String error) {
39     Nodo nuevo;
40     if( vacia() ) {
41         principal = new Nodo(error);
42     }else {
43         nuevo = new Nodo(error);
44         nuevo.sig = principal;
45         principal = nuevo;
46     }
47 }
48 }
49 public void limpiar() {
50     principal = null;
51 }

53 public boolean vacia()
54 {
55     return (principal == null)? true:false;
56 }
57 class Nodo
58 {
59     private String error;
60     Nodo sig;
61     public Nodo()
```

PilaErrores.java

Source	History
--------	---------

```
47      }
48    }
49    public void limpiar() {
50      principal = null;
51    }
52
53    public boolean vacia()
54    {
55      return (principal == null) ? true:false;
56    }
57    class Nodo
58    {
59      private String error;
60      Nodo sig;
61      public Nodo()
62      {
63        sig = null;
64        error = "";
65      }
66      public Nodo(String error)
67      {
68        this.error = error;
69        sig = null;
70      }
71    }
72  }
73 }
```

The screenshot shows a Java code editor window with the following details:

- Title Bar:** TablaSimbolos.java
- Toolbar:** Source, History, and various navigation icons.
- Code Area:** Displays the source code for the `TablaSimbolos` class.

```
1 package utils;
2
3 import javax.swing.table.DefaultTableModel;
4
5 public class TablaSimbolos {
6
7     private Nodo raiz;
8     private Nodo fin;
9     private Nodo rec;
10    private DefaultTableModel modelo;
11    private int contador = 0;
12    public TablaSimbolos() {
13        raiz = fin = null;
14    }
15
16    public void insertar(Token token) {
17        Nodo nuevo = new Nodo(token);
18        contador++;
19        if (vacia()) {
20
21            raiz = fin = nuevo;
22        } else {
23            fin.sig = nuevo;
24            fin = nuevo;
25        }
26    }
27
28    public void recorrer() {
29        Nodo rec = raiz;
30        while (rec != null) {
31            System.out.println(rec.token.getLexema());
32        }
33    }
34}
```

TablaSimbolos.java

Source	History															
--------	---------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

```
30     while (rec != null) {
31         System.out.println(rec.token.getLexema());
32         rec = rec.sig;
33     }
34 }
35
36 [-] public void limpiar() {
37     contador = 0;
38     raiz = fin = rec = null;
39 }
40
41 [-] public boolean vacia() {
42     return (raiz == null) ? true : false;
43 }
44
45 public Token pop()
46 {
47     if(rec == null)
48         rec = raiz;
49     Token token = rec.token;
50     rec = rec.sig;
51     return token;
52 }
53
54
55 public Token[] toArray()
56 {
57     Token []tokens = new Token[contador];
58     int i = 0;
59     Nodo rec = raiz;
60     while(rec != null)
```



TablaSimbolos.java

```
Source History |               
```

74       public Token buscar(String lexema) {  
75            return buscar(0, lexema);  
76       }  
77       public void llenarModelo(DefaultTableModel modelo) {  
78            this.modelo = modelo;  
79            llenarModelo();  
80       }  
81       private void limpiarModelo()  
82       {  
83            modelo.setRowCount(0);  
84       }  
85       public void llenarModelo() {  
86            limpiarModelo();  
87            Nodo rec = raiz;  
88            while(rec != null) {  
89                modelo.addRow(rec.token.datos());  
90                rec = rec.sig;  
91            }  
92       }  
93  
94       public Token buscar(int etiqueta, String lexema) {  
95            Nodo rec = raiz;  
96            Token token = null;  
97  
98            while (rec != null)  
99                if (rec.id == etiqueta || rec.token.getLexema().equals(lexema))  
100                    return rec.token;  
101  
102            return token;  
103       }  
104       public void setModelo(DefaultTableModel modelo) {

TablaSimbolos.java

```
Source History
public Token buscar(int etiqueta, String lexema) {
    Nodo rec = raiz;
    Token token = null;

    while (rec != null)
        if (rec.id == etiqueta || rec.token.getLexema().equals(lexema))
            return rec.token;

    return token;
}

public void setModelo(DefaultTableModel modelo) {
    if(this.modelo == null)
        this.modelo = modelo;
}

class Nodo {
    Token token;
    int id;
    Nodo sig;

    public Nodo(Token token) {
        sig = null;
        id = token.getEtiqueta();
        this.token = token;
    }
}
```

Token.java

```
Source History
package utils;

import lexico.Estados;

public class Token{
    private int etiqueta;
    private int id;
    private String lexema;
    private String descripcion;
    private int linea, posicion;

    public Token() {}
    public Token(int etiqueta) {
        this.etiqueta = etiqueta;
    }
    public Token(int etiqueta, String lexema) {
        this.etiqueta = etiqueta;
        this.lexema = lexema;
    }
    public Token(int etiqueta, String lexema, String descripcion) {
        this.etiqueta = etiqueta;
        this.lexema = lexema;
        this.descripcion = descripcion;
    }
    public Token(int etiqueta, String lexema, String descripcion, int linea, int posicion) {
        super();
        this.etiqueta = etiqueta;
        this.lexema = lexema;
        this.descripcion = descripcion;
        this.linea = linea;
    }
}
```

The screenshot shows a Java code editor window titled "Token.java". The code is a class definition for "Token". The editor interface includes a toolbar at the top with various icons for file operations like Open, Save, and Print. Below the toolbar is a menu bar with "Source" and "History" tabs. The main area displays the following Java code:

```
30     this.descripcion = descripcion;
31     this.linea = linea;
32     this.posicion = posicion;
33 }
34
35
36 public Token(int etiqueta, int id, String lexema, String descripcion, int linea, int posicion) {
37     super();
38     this.etiqueta = etiqueta;
39     this.id = id;
40     this.lexema = lexema;
41     this.descripcion = descripcion;
42     this.linea = linea;
43     this.posicion = posicion;
44 }
45 public boolean es(Estados e)
46 {
47     return (e.getEtiqueta() == this.etiqueta);
48 }
49
50 public int getEtiqueta() {
51     return etiqueta;
52 }
53 public void setEtiqueta(int etiqueta) {
54     this.etiqueta = etiqueta;
55 }
56 public String getLexema() {
57     return lexema;
58 }
59 public void setLexema(String lexema) {
60     this.lexema = lexema;
61 }
```

Token.java

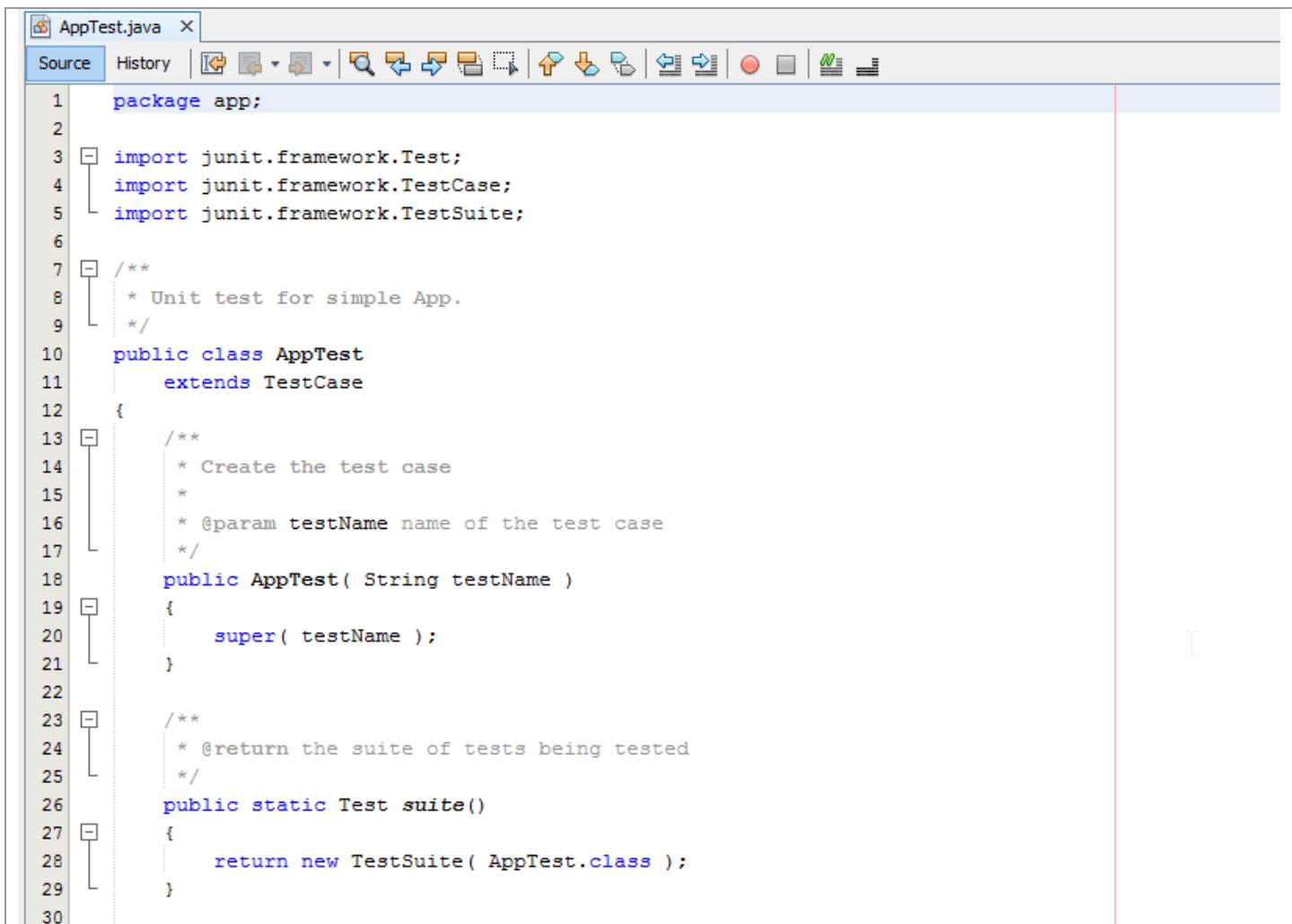
Source History

```
62     public String getDescripcion() {
63         return descripcion;
64     }
65     public void setDescripcion(String descripcion) {
66         this.descripcion = descripcion;
67     }
68     public String[] datos()
69     {
70         String ID = (id == 0) ? "":id+"";
71         String []datos = {etiqueta+"",ID,lexema,descripcion};
72         return datos;
73     }
74     public int getLinea() {
75         return linea;
76     }
77     public void setLinea(int linea) {
78         this.linea = linea;
79     }
80     public int getPosicion() {
81         return posicion;
82     }
83     public void setPosicion(int posicion) {
84         this.posicion = posicion;
85     }
86     public int getId() {
87         return id;
88     }
89     public void setId(int id) {
90         this.id = id;
91     }
92 }
```

Token.java

Source History

```
72     return datos;
73 }
74 public int getLinea() {
75     return linea;
76 }
77 public void setLinea(int linea) {
78     this.linea = linea;
79 }
80 public int getPosicion() {
81     return posicion;
82 }
83 public void setPosicion(int posicion) {
84     this.posicion = posicion;
85 }
86 public int getId() {
87     return id;
88 }
89 public void setId(int id) {
90     this.id = id;
91 }
92
93
94 }
95 }
```



The screenshot shows a Java code editor window with the file `AppTest.java` open. The code implements a JUnit test case for an application. It includes imports for `junit.framework.Test`, `junit.framework.TestCase`, and `junit.framework.TestSuite`. The class `AppTest` extends `TestCase` and provides a constructor that takes a `String` parameter named `testName`. It also defines a static method `suite()` which returns a new `TestSuite` containing the current class. The code uses JavaDoc-style comments to describe the methods.

```
1 package app;
2
3 import junit.framework.Test;
4 import junit.framework.TestCase;
5 import junit.framework.TestSuite;
6
7 /**
8 * Unit test for simple App.
9 */
10 public class AppTest
11     extends TestCase
12 {
13     /**
14      * Create the test case
15      *
16      * @param testName name of the test case
17      */
18     public AppTest( String testName )
19     {
20         super( testName );
21     }
22
23     /**
24      * @return the suite of tests being tested
25      */
26     public static Test suite()
27     {
28         return new TestSuite( AppTest.class );
29     }
30 }
```

AppTest.java

Source History |

```
19     {
20         super( testName );
21     }
22
23     /**
24      * @return the suite of tests being tested
25      */
26     public static Test suite()
27     {
28         return new TestSuite( AppTest.class );
29     }
30
31     /**
32      * Rigourous Test :-)
33      */
34     public void testApp()
35     {
36         assertTrue( true );
37     }
38 }
39 }
```

The screenshot shows a Java code editor window with the file `TestLexico.java` open. The code is a test class for a lexical analyzer. It imports various Java and JUnit packages, including `Assertions.assertEquals`, `File`, `IOException`, `Test`, `Lexico`, `PilaErrores`, and `TablaSimbolos`. The class contains two test methods: `testLectura()` and `testPalabrasReservadas()`. The code uses reflection to access private fields of the `Lexico` class. Error markers are present on several lines, indicating compilation issues.

```
1 package app;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
5 import java.io.File;
6 import java.io.IOException;
7
8 import org.junit.jupiter.api.Test;
9
10 import lexico.Lexico;
11 import utils.PilaErrores;
12 import utils.TablaSimbolos;
13
14 class TestLexico {
15
16     @Test
17     void testLectura() throws IOException {
18         File archivo = new File("palabras");
19         Lexico lexico = new Lexico(archivo);
20         char caracter = ' ';
21         while(caracter != '\u0000')
22         {
23             caracter = lexico.obtenerCaracter();
24             //System.out.print(caracter);
25         }
26     }
27
28     @Test
29     void testPalabrasReservadas() throws IOException
30     {
31     }
32 }
```

The screenshot shows a Java code editor with the file `TestLexico.java` open. The code is a test method for a lexical analyzer. Several parts of the code are underlined with red, indicating syntax errors or warnings:

```
24     ..... //System.out.print(caracter);
25     ..... }
26 }
27
28
29 @Test
30 void testPalabrasReservadas() throws IOException
31 {
32     File archivo = new File("palabras");
33     PilaErrores pilaErrores = new PilaErrores();
34     boolean real;
35     TablaSimbolos tabSimbolos = new TablaSimbolos();
36     Lexico lexico = new Lexico(archivo,pilaErrores,tabSimbolos);
37     real = lexico.analizar();
38     tabSimbolos.recorrer();
39     while(!pilaErrores.vacia())
40     {
41         System.out.println(pilaErrores.pop());
42     }
43     assertEquals(true, real);
44 }
45
46
47 }
```

Annotations in the code editor:

- Line 29: `@Test` is underlined in red.
- Line 43: `assertEquals(true, real);` is underlined in red.
- Line 44: A yellow warning icon is shown next to the closing brace of the test method.
- Line 45: A red error icon is shown next to the first character of the blank line.

```
1 package app;
2
3 import static org.junit.Assert.fail;
4
5 import java.io.File;
6 import java.io.IOException;
7
8 import org.junit.jupiter.api.Test;
9
10 import lexico.Lexico;
11 import sintactico.Sintactico;
12 import utils.PilaErrores;
13 import utils.TablaSimbolos;
14 import utils.Token;
15
16 class TestSintactico {
17
18
19     @Test
20     void testSintactico() throws IOException
21     {
22         File archivo = new File("testSintactico");
23         Lexico l = new Lexico(archivo);
24         l.analizar();
25         TablaSimbolos tab = l.getTabSimbolos();
26         PilaErrores pila = new PilaErrores();
27         Sintactico s = new Sintactico(pila);
28         Token simbolos[] = tab.toArray();
29         s.setSimbolos(simbolos);
30         boolean real = s.analizar();
```

The screenshot shows a Java code editor window titled "TestSintactico.java". The code is a test method for a syntactic analyzer. It imports several classes: File, Lexico, TablaSimbolos, PilaErrores, Sintactico, and Token. The code initializes these objects, sets up symbol tables, and performs a syntax analysis. It then checks if the analysis was successful by popping errors from a stack and failing the test if any errors remain.

```
20  @Test
21  void testSintactico() throws IOException
22  {
23      File archivo = new File("testSintactico");
24      Lexico l = new Lexico(archivo);
25      l.analizar();
26      TablaSimbolos tab = l.getTabSimbolos();
27      PilaErrores pila = new PilaErrores();
28      Sintactico s = new Sintactico(pila);
29      Token simbolos[] = tab.toArray();
30      s.setSimbolos(simbolos);
31      boolean real = s.analizar();
32
33      if(!real)
34      {
35          while(!pila.vacia())
36              System.out.println(pila.pop());
37          fail("Analisis incorrecto");
38      }
39
40
41
42
43
44 }
45
```

The screenshot shows a Java IDE interface with two code editors for the same file, `TestTablaSimbolos.java`.

**Top Editor (Lines 1-31):**

```
1 package app;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
5 import org.junit.jupiter.api.Test;
6
7 import utils.TablaSimbolos;
8 import utils.Token;
9
10 class TestTablaSimbolos {
11
12
13     void testInsertar() {
14         TablaSimbolos tabla = new TablaSimbolos();
15         tabla.insertar(new Token(10));
16         tabla.insertar(new Token(11));
17         tabla.insertar(new Token(12));
18         tabla.insertar(new Token(13));
19         tabla.insertar(new Token(14));
20         tabla.recorrer();
21     }
22     @Test
23     void testToArray()
24     {
25         TablaSimbolos tabla = new TablaSimbolos();
26         tabla.insertar(new Token(10));
27         tabla.insertar(new Token(12));
28         tabla.insertar(new Token(13));
29         tabla.insertar(new Token(14));
30         Token esperado[] = {new Token(10),new Token(12),new Token(13),new Token(14)};
31         Token real[] = tabla.toArray();
32     }
33 }
```

**Bottom Editor (Lines 30-50):**

```
30     Token esperado[] = {new Token(10),new Token(12),new Token(13),new Token(14)};
31     Token real[] = tabla.toArray();
32     for(int i = 0 ; i < 4; i++)
33         assertEquals(esperado[i].getEtiqueta(),real[i].getEtiqueta());
34     }
35     @Test
36     void testPop()
37     {
38         TablaSimbolos tabla = new TablaSimbolos();
39         tabla.insertar(new Token(10));
40         tabla.insertar(new Token(12));
41         tabla.insertar(new Token(13));
42         tabla.insertar(new Token(14));
43         assertEquals(10, tabla.pop().getEtiqueta());
44         assertEquals(12, tabla.pop().getEtiqueta());
45         assertEquals(13, tabla.pop().getEtiqueta());
46         assertEquals(14, tabla.pop().getEtiqueta());
47     }
48
49 }
50 }
```

Udb.edu.sv

<http://www.udb.edu.sv/udb/archivo/guia/informatica-ingenieria/compiladores/2013/ii/guia-3.pdf>

Alfred V Aho  
1990  
Compiladores  
Argentina [etc]  
Addison-Wesley Iberoamerican

pro  
ANALIZADOR SINTACTICO: INTRODUACION, CONCEPTOS, CARACTERISTICAS  
Es.slideshare.net

<https://es.slideshare.net/Infomaniapro/analizador-sintactico-introducion-conceptos-caracteristicas>

Analizador sintáctico LL – EcuRed  
Ecured.cu  
[https://www.ecured.cu/Analizador\\_sint%C3%A1ctico\\_LL](https://www.ecured.cu/Analizador_sint%C3%A1ctico_LL)

Analizador sintáctico LR – EcuRed  
Ecured.cu  
[https://www.ecured.cu/Analizador\\_sint%C3%A1ctico\\_LR](https://www.ecured.cu/Analizador_sint%C3%A1ctico_LR)

## Referencias

[http://space.wccnet.edu/~pmillis/cps120/cps120\\_pgm\\_syntax.pdf](http://space.wccnet.edu/~pmillis/cps120/cps120_pgm_syntax.pdf)

Cidecame.uaeh.edu.mx. (2018). 3.1 *Funcion del Analizador Sintáctico*. [online] Available at:  
[http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/autocontenido/autocon/31\\_funcion\\_del\\_analizador\\_sintctico.html](http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/autocontenido/autocon/31_funcion_del_analizador_sintctico.html).

Dsc.itmorelia.edu.mx. (2018). [online] Available at:  
[http://dsc.itmorelia.edu.mx/~jcolivares/courses/ps207a/ps2\\_u4.pdf](http://dsc.itmorelia.edu.mx/~jcolivares/courses/ps207a/ps2_u4.pdf)

Lcc.uma.es. (2018). [online] Available at: <http://www.lcc.uma.es/~galvez/ftp/tci/tictema3.pdf>