MANUAL TECNICO

En el presente proyecto se hicieron uso de diferentes aplicaciones que fueron utilizadas para el desarrollo de la aplicación siendo estas:

- Netbeans
- Graphviz

Netbeans

Fue el programa que utilizamos para el desarrollo tanto del fronted como del backend, la versión de Netbeans es la 12.6. Para el desarrollo de la aplicación se trabajó todo en Maven, además de utilizar la librería swing para el desarrollo del fronted.

Java

La versión de java utilizada para el desarrollo del proyecto fue la 15.6

Graphviz

Fue el programa que utilizamos para la creación de las imágenes que nos solicitaba el proyecto, los pasos que se necesitan para utilizar graphviz son:

- Descargar la última versión de Graphviz desde tu navegador
- Hacer el proceso de Instalación
- Ir al Panel de Control y agregar la dirección de Graphviz al espacio de variables
- Hacer una prueba en consola y listo

Luego de realizar estos pasos solo quedaría hacer la conexión entre netbeans y graphviz con el objetivo de poder crear las imágenes sin necesidad de usar la terminal y luego se eso podemos seguir con el desarrollo del programa.

Pasando a la creación del programa lo primero que hicimos fue definir nuestro alfabeto que sería utilizado durante la ejecución del programa y con el cual iba a ser posible diferenciar los diferentes tokens que el usuario nos mandará.

Siendo esa lista los siguientes

A continuación, hablaremos de cada una de las clases que conforman **fronted** del código y sus respectivas funciones:

Analizador

Esta es la clase principal del programa ya que desde aquí se crear las ramas o conexiones con otras clases para el funcionamiento del programa y dentro de esta clase podemos encontrar los siguientes métodos, el sistema se divide en dos partes aunque forman parte del mismo fronted:

cargarArchivos(): Este método es utilizado para la cargar de archivos de texto que el usuario tenga guardado en algún lugar de su computadora y al momento de presionar esta opción se debe verificar que el path ingresado exista y que el archivo se pueda leer de lo contrario notifica al usuario para que haga los ajusten necesarios para su uso, además de que limpia todo el panel del editor para reemplazarlo por el nuevo archivo de entrada. Luego de esto utiliza el método crearVisualizacionDeArchivo(): de la clase Archivo que retornara una lista la cual tendrá dentro la cadena de caracteres del archivo y para mostrarlo en pantalla se invocara el método de mostrarArchivo() que se detalla más adelante.

visualizarGrafica(): Este método es utilizado para llamar el panel Grafica que se detallará más adelante, pero antes verifica que haya tokens que visualizar.

activarReconocimientoDeToken(): Este método es utilizado para mandar todo el texto presentado en pantalla a una evaluación para determinar cada tokens existente, pero primero verifica que el panel tenga información dentro de lo contrario le notificara al usuario que debe ingresar información al panel para ser procesada, luego de esto invoca al método de organizarCadena() de la clase Archivo. Si todo esta bien con los tokens entonces pasa a la siguiente sección que es la del analizador sintáctico llamando al método analizarListaDeTokens() de la Clase Analizador de Tokens.

consulta(): Este método es utilizado para llamar al panel Ayuda que se detallará más adelante.

información(): Este método es utilizado para llamar al panel AcercaDe que se detallará más adelante.

visualizar(): Este método es utilizado para llamar al panel de Reportes que se detallara más adelante, pero antes verifica que haya tokens que visualizar.

mostrarErrores(): Este método es utilizado para mostrar en el panel de errores todos los tokens que no desechados o mal redactados para que el usuario se percate de su error y haga las correcciones necesarias

mostrarTokens(): Este método muestra todos los tokens que el sistema encontró que concidian con los parámetros solicitados, además de que invoca al método verColores() con el fin de colorear las letras y el usuario pueda ver con mayor información los elementos que conforman el código.

verColores(): Este método es utilizado para ponerle color a cada token encontrado y así tener una mejor vista del código.

obtenerTextoEscrito(): Este método obtiene todo lo que esta escrito en el panel y lo guarda para su posterior análisis.

Grafica

En esta clase es donde se va poder visualizar todos los tokens encontrados de una manera más detallada y representándolos por medio de graficas de estados. Los métodos utilizados fueron los siguientes, que están divididos en dos partes que son parte del mismo fronted:

Como primer punto a tomar en cuenta hay que destacar que los siguientes métodos solo son utilizados cuando el usuario oprime un botón según sea el token que desee visualizar, y cada método invoca al método mostrarLista(): y solo cambio el tipo de token por el que se desea buscar:

- Identificadores()
- Aritmeticos()
- Comparacion()
- Asignacion()
- PalabrasClave()
- Constante()
- Comentario()
- Otros()

mostrarLista(): Este método es utilizado para mostrar en un panel la lista de token que cumplen con ser parte de la misma categoría seleccionada y los clasifica según su posición para que sea más legible de leer.

activarGrafica(): Este método es utilizado para mostrar la gráfica del token pero primero manda a llamar a la clase CrearImagen y le mando la información necesaria luego busca la imagen en el path local y muestra el resultado en pantalla junto con una pequeña descripción del token y el tipo de token que es.

seleccionar(): Este método es utilizado para asegurar la selección del token y desactivar su interacción, además de que analiza que el numero este entre el rango de números establecido en el apartado de mostrarLista().

obtenerNumeroDeToken(): Este método es utilizado para obtener el número del token que el usuario quiere visualizar

Reporte

En esta clase se podrá visualizar un reporte detallado de cada tokens encontrado, así como la opción para obtener la información de cada bloque o línea de codigo, también está divido en dos partes que son parte del fronted.

mostrarInformacionTokens(): Este método redirecciona la búsqueda desea hacia el buscador correcto dependiendo de si se debe mostrar un bloque o solo una línea.

mostrarTablaDeSimbolos(): Este método muestra todos los tokens por separado junto con su fila, su columna, su lexema y su patrón.

obtenerColumnas(): Este método sirve para crear las columnas que se van a mostrar en el reporte.

mostrarEspecificos(): Muestra solo la información superficial de un bloque, es decir solo una línea tanto para los bloques como para las variables y otros.

mostrarInformacionBloque(): Este método crea las columnas, si en caso se selecciona la opción de variables y otras se crear una columna extra de invocaciones en las cual se ha de mostrar las veces que el método fue invocado durante todo el código.

mostrarBloque(): Este método muestra el bloque que el usuario selecciono y el modo de mostrarlo es tomar el valor if, while, for u otro como punto de partida y si algún token tiene esta en una columna menor a esta se rompe el ciclo y se muestra los elementos guardados.

obtenerNumeroDeInvocaciones(): Este método cuenta todas las veces que un método es llamado durante toda la ejecución del programa.

regresar(): Este método es utilizado regresar al panel del Analizador.

mostrarTabla(): Este método muestra la tabla con todos los datos relevantes de cada token como son su tipo, patrón, lexema, fila y columna.

Reportes de Bloque

En esta clase se podrá visualizar bloques de código que el usuario elija como puede ser un bloque if, un while, un for, entre otros, y se mostrara todos los bloque de ese campo que estén presentes dentro del código.

mostrarBloques(): Este método toma como parámetro la opción que el usuario selecciono en el fronted y hace un análisis parar determinar que clase de bloque es el que debe mostrar, asi como determinar si la búsqueda se debe hacer pensando en un bloque o si solo se trata de una línea.

analizarBloque(): Este método toma como referencia al if, while, for encontrado para realizar un análisis y mostrar solo los elementos que se encuentren en la misma columna o que sea mayor al valor de referencia de lo contrario se romperá el ciclo y mostrara todo lo que haya guardado con anterioridad.

colocarTexto(): Este método colocara el texto en el panel para que el usuario lo pueda visualizar.

Acerca de

Esta clase solo muestra la información del programador.

Ayuda

Esta clase muestra una pequeña guía que puede consultar en caso tenga alguna duda del funcionamiento del programa.

A continuación, hablaremos de cada una de las clases que conforman **backend** del código y sus respectivas funciones:

Archivo

En esta clase se va abstraer la información del fichero seleccionado, además de que se hará un primer análisis al texto seleccionado por el usuario.

crearVisualizacionDeArchivos(): Este método es utilizado para obtener la información del fichero seleccionado previamente por el usuario en el apartado del analizado. Y cada línea obtenida lo agrega a nuestra lista para su posterior análisis.

organizarCadena(): Este método es invocado cuando el usuario presiona el botón de activarReconocimientoDeTokens() y manda toda la cadena y en este método hace un pequeño análisis inicial en busca de espacios en blanco, comillas, numerales o fines de líneas con el objetivo de crear tokens que luego serán analizados en la clase Crear Tokens.

Crear Token

En esta clase se realizará un análisis más detallado de cada tokens que se divido en la sección anterior en la clase Archivo método organizarCadena() y aquí se obtendrán la definión correcta de cada token.

analizarComentarios(): Este método es utilizado cuando en el método organizarCadena() de la clase Archivo se encuentra un # por lo tanto manda todo el cargo de esa fila para que este método ubique el final de la línea y cree el tokens.

analizarCentral(): Este método es utilizado cuando en el método organizarCadena() de la clase Archivo se encontró un espacio o un salto de línea y manda lo que haya concatenado a este método que revisara si se trata de un símbolo de la tabla especificado, una constante, una combinación de tokens, un identificador o un error.

analizarTipoDeTokens(): Este método es utilizado para verificar si la palabra encontrada tiene relación con algún valor de nuestra tabla de símbolos definida de forma predeterminada. De igual manera lleva un valor booleano en caso no se quiera crear el token y el motivo es para mantener el orden de cadena y que no hay errores de escritura.

analizarConstante(): Este método es utilizado para determinar si la cadena que tenemos en ese momento cumple con las características para ser determinada como una constante, de lo contrario retorna un false.

analizarRestante(): Este método es utilizado para hacer una revisión final al código solo que esta vez buscando palabras o signos de nuestra tabla de símbolos para así determinar tokens, acabe destacar que este método implementa a los dos métodos anteriores en su estructura.

analizarSigno(): Este método es utilizado para determinar si el signo que encontramos no es parte de otro compuesto, y si lo fuera se concatenan los dos datos y se manda a realizar el token.

analizarIdentificadores(): Este método es el último en donde se evaluara los datos concatenados y si no cumple con el modo de escritura de un identificador entonces se creara un error que será notificado al usuario en la pantalla principal de lo contrario se guardara como un token y ahí el proceso se repetirá hasta terminar de leer todo el panel de texto.

analizarCadena(): Este es un método especial dado de que solo será invocado en caso se encuentra una comilla simple o doble en la cadena de caracteres que se esté leyendo, y este análisis abarca desde el punto donde se encontró las comillas hasta el final de la fila y si llega al final y no encuentra la otra comilla se marcara como error, de lo contrario se guardara la cadena y se devolverá el valor de la columna correspondiente a leer.

saberFinDeLinea(): Este método es utilizado para saber si al final de verificar la cadena existe un salto de línea para determinar si se debe reiniciar el contador de la columna o si sigue con el mismo número obtenido.

Crear Imagen

Esta clase es utilizada para la creación de imágenes que serán mostradas en el apartado de graficas. Y los métodos que utilizamos son los siguientes:

generarImagen(): Este método es utilizado para generar la imagen que se mostrará en la aplicación y para eso ejecuta por medio de los métodos determinados la orden como simulando la terminal para la creación del imagen.

crearArchivo(): Este método es utilizado para guardar la información que tenemos en el documento dot donde se guarda la información de la gráfica que luego será ejecutada para la creación de la imagen.

obtenerContenido(): Este método es utilizado para obtener el código necesario para la creación de la imagen y lo guarda todo en el archivo predefinido.

Analizador de Tokens

Esta clase es utilizada para analizar los tokens que nos manda el analizador léxico y determinar que la estructura que nosotros estamos buscando o que el usuario trata de ingresar es la correcta de lo contrario mandara un mensaje de error junto con los posibles signos o datos que se esperaban con la idea de que el usuario tenga una idea más clara de donde esta su error y lo que tiene que hacer para arreglarlo.

analizarListaDeTokens(): Este método es el que recibe todos los tokens y es el que administra cada los para determinar si se pasa a examinar al otro bloque o si se regresa al principio para seguir analizando el siguiente tokens.

verificarAsignacionOMetodo(): Este método es utilizado para determinar si el conjunto de tokens que tenemos forman una asignación en las n formas que están establecidas en el método y si existiera un error se crea el error y se le mandara un mensaje al usuario.

analizarAgrupaciones(): Este método es utilizado para analizar que una parte del código cuando los tokens entrantes tengan la forma (a,b,c) ó [a:b:c]

analizarAsignacionEspecial(): Este método analiza los tokens en caso venga de la forma a,b = c, d

verificarDef(): Este método analiza los tokens para determinar si se trata de un método en la n formas que están definidas dentro del programa de lo contrario manda un mensaje de error al usuario con sus posibles soluciones.

verificarIf(): Este método analiza los tokens para determinar si se trata de una sentencia if en la n formas que están definidas dentro del programa de lo contrario manda un mensaje de error al usuario con sus posibles soluciones.

verificarWhile(): Este método analiza los tokens para determinar si se trata de una while en las n formas que están definidas dentro del programa de lo contrario manda un mensaje de error al usuario con sus posibles soluciones.

verificarFor(): Este método analiza los tokens para determinar si se trata de una for en las n formas que están definidas dentro del programa de lo contrario manda un mensaje de error al usuario con sus posibles soluciones.

estaEnLaMismaLinea(): Este método verifica las instrucciones estén en la misma línea de código sin bajar a la siguiente de lo contrario se marca como un error.

imprimiError(): Si existe un error lo imprime y lo manda al usuario.

Tabla de Símbolos

Esta clase es utilizada para definir los símbolos y expresiones validas que se pueden tener una cadena para que un tokens sea aceptado, estos valores son fijos durante toda la construcción y ejecución del programa. Y obtenemos todo este arreglo por medio de un método obtener para utilizarlo durante la ejecución.

Token

Esta clase es la que guarda toda la información acerca de un token que hayamos encontrado y lo almacena para su utilización cuando sea necesario.

obtenerTipoDeToken(): Este método es utilizado para mandar el tipo de token de la cadena.

obtenerLexema(): Este método es utilizado para mandar la palabra que guardamos como lexema que ya fue analizada anteriormente.

obtenerFila(): Con este método obtenemos la fila donde esta ubicado el token.

obtenerColumna(): Con este método obtenemos la columna donde está ubicado el token.

obtenerPatron(): Con este método obtenermos el patrón con el cual el token fue valido.

agregarPatron(): Con este método definimos el patrón que tendrá nuestro token en base a al información que nos da el analizador.

Lista Elementos

Se hizo uso de las listas de elementos para guardar los tokens.

agregarALaLista(): Este método es utilizado para agregar un nuevo token que hayamos encontrado y como no sabemos cuántos tokens vamos a recibir es por eso que se utiliza una lista dinámica para poder tener un mejor control y no preocuparnos por el espacio.

eliminar(): Este método elimina el nodo en la posición que sea solicitado.

encontrarPorIndice(): Este método nos permite ubicar el token que deseamos visualizar.

obtenerContenido(): Con este método obtenemos el contenido que tenemos guardado en ese nodo.

estaVacia(): Verifica si esta vacía y si esta devuelve un valor true.

getLongitud(): Este método es utilizado para determinar el largo de una lista.

Tabla de símbolos

A continuación, presentamos la tabla de símbolos que son las normas con las cuales basamos el analizador léxico de este programa, además, tomando como punto de partida las restricciones que nos presentaba asimilar una estructura al lenguaje de programación Python. Pero por medio de una separación de caracteres su pudo crear la siguiente tabla de dominio.

Tabla de Simbolos	
Tipo de Palabra	Patron
Indentificador	([\w]|_)+(\w|\d)*
Aritmeticos	suma, resta, mult, div
Compararción	igual, mayor que, menor que
Asignación	Para la asignación de variables
Palabras Clave	while, and, if, else, def
Numeros	(0-9)+(.|0-9)(0-9)+
Otro	(,), {, }, [,],

Para finalizar presentamos el diagrama de clases que utilizamos para la resolución y creación del programa:

Producciones

A continuación presentamos las cadenas de producción para cada caso:

Asignación:

S -> I=P

P -> IP

| (Expresion Agrupada)

| ifP

| notP

| elseP

| e

If:

S -> ifR

R -> BR

| IG

| notR

| :

B -> True

| False

G -> (Signo de Comparación)R

For:

S -> forW

W -> IX

| key, value in I.items():

X -> inIL

L -> :

| (Expresion Agrupada)

While:

S -> while IF

F -> (Signo de Comparación)F

| (Expresion Agrupada)F

| (Signo Aritmetico)F

| IF

| :

Print:

S -> print(N

N -> IN

| fJ

|)

| I,N

| I.I())

J -> Constante)

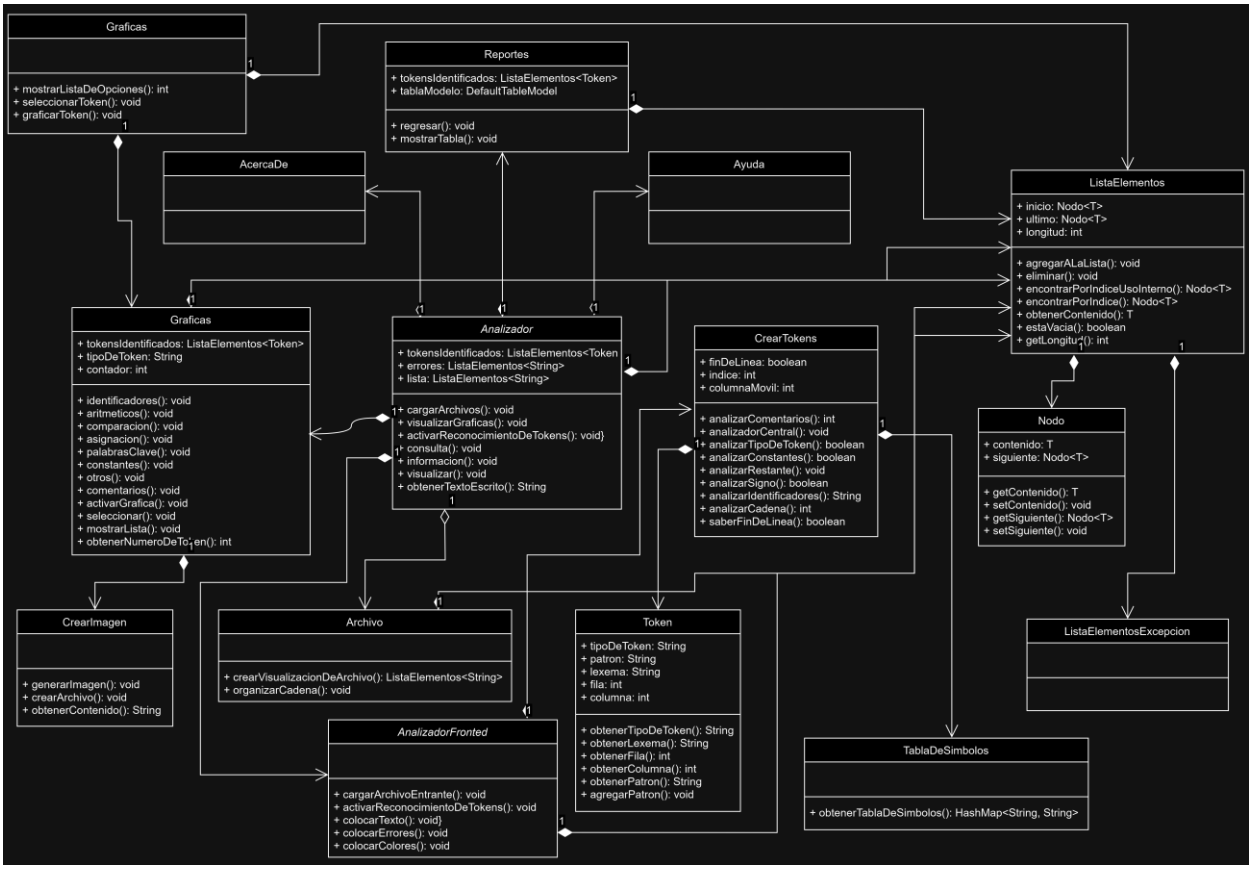
Comentario:

S -> #K

K -> LetraK

| DigitoK

| e


Graficas
+ mostrarListaDeOpciones(): int
+ seleccionarToken(): void
+ graficarToken(): void
Reportes
+ tokensIdentificados: ListaElementos<Token>
+ tablaModelo: DefaultTableModel
+ regresar(): void
+ mostrarTabla(): void
AcercaDe
Ayuda
ListaElementos
+ inicio: Nodo<T>
+ ultimo: Nodo<T>
+ longitud: int
+ agregarALaLista(): void
+ eliminar(): void
+ encontrarPorIndiceUsoInterno(): Nodo<T>
+ encontrarPorIndice(): Nodo<T>
+ obtenerContenido(): T
+ estaVacia(): boolean
+ getLongitu(): int
Graficas
+ tokensIdentificados: ListaElementos<Token>
+ tipoDeToken: String
+ contador: int
+ identificadores(): void
+ aritmeticos(): void
+ comparacion(): void
+ asignacion(): void
+ palabrasClave(): void
+ constantes(): void
+ otros(): void
+ comentarios(): void
+ activarGrafica(): void
+ seleccionar(): void
+ mostrarLista(): void
+ obtenerNumeroDeTo¡en(): int
Analizador
+ tokensIdentificados: ListaElementos<Token
+ errores: ListaElementos<String>
+ lista: ListaElementos<String>
+ cargarArchivos(): void
+ visualizarGraficas(): void
+ activarReconocimientoDeTokens(): void}
+ consulta(): void
+ informacion(): void
+ visualizar(): void
+ obtenerTextoEscrito(): String
CrearTokens
+ finDeLinea: boolean
+ indice: int
+ columnaMovil: int
+ analizarComentarios(): int
+ analizadorCentral(): void
+ analizarTipoDeToken(): boolean
+ analizarConstantes(): boolean
+ analizarRestante(): void
+ analizarSigno(): boolean
+ analizarIdentificadores(): String
+ analizarCadena(): int
+ saberFinDeLinea(): boolean
Nodo
+ contenido: T
+ siguiente: Nodo<T>
+ getContenido(): T
+ setContenido(): void
+ getSiguiente(): Nodo<T>
+ setSiguiente(): void
CrearImagen
+ generarImagen(): void
+ crearArchivo(): void
+ obtenerContenido(): String
Archivo
+ crearVisualizacionDeArchivo(): ListaElementos<String>
+ organizarCadena(): void
Token
+ tipoDeToken: String
+ patron: String
+ lexema: String
+ fila: int
+ columna: int
+ obtenerTipoDeToken(): String
+ obtenerLexema(): String
+ obtenerFila(): int
+ obtenerColumna(): int
+ obtenerPatron(): String
+ agregarPatron(): void
ListaElementosExcepcion
AnalizadorFronted
+ cargarArchivoEntrante(): void
+ activarReconocimientoDeTokens(): void
+ colocarTexto(): void}
+ colocarErrores(): void
+ colocarColores(): void
TablaDeSimbolos
+ obtenerTablaDeSimbolos(): HashMap<String, String>