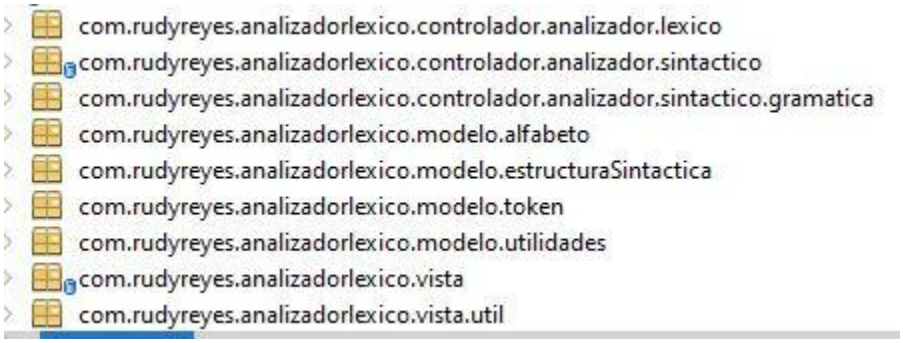
MANUAL TÉCNICO

1. Introducción y Propósito:

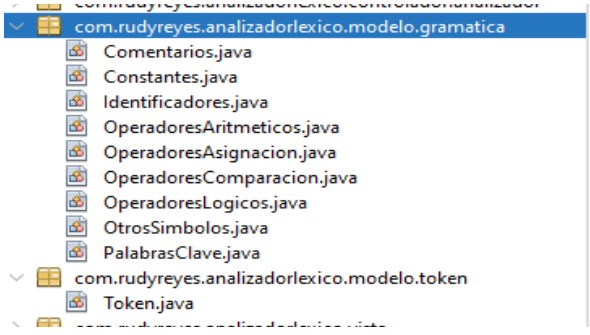
El siguiente proyecto contiene un software desarrollado en JAVA 17 cuya función es ser un analizador léxico basado en Python, el programa solamente analizará el código ingresado por el usuario y clasificará los tokens encontrados en el código.

2. Arquitectura y Diseño:

En el programa desarrollado se aplicó el diseño MVC (Modelo, Vista, Controlador), donde en el Modelo se puede observar el alfabeto propuesto, la clase Token y la clase EstructuraSintactica que permitirá generar los objetos del tipo token, para clasificarlos, en el Controlador se encuentra la clase AnalizadorLexico, AnalizadorSintactico aquí es donde se enviará el código ingresado por el usuario para ser analizado, esta clase clasificará cada Token, haciendo uso de las clases del Modelo, luego retornará una lista de tokens para que la Vista los maneje y muestre los resultados al usuario.



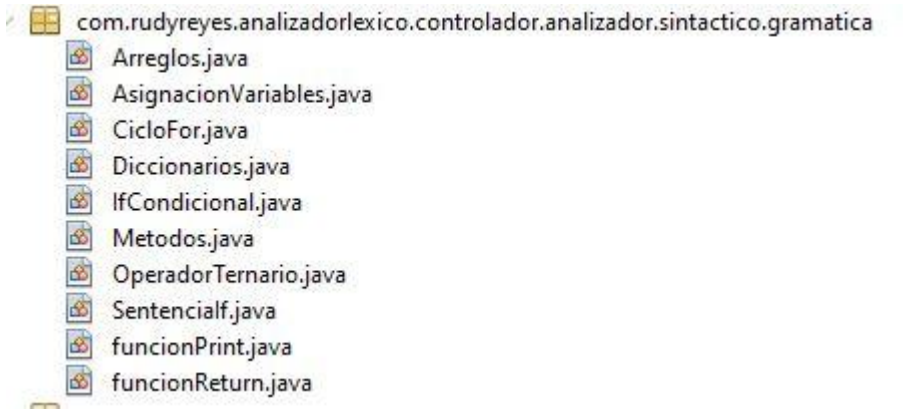
Modelo:



Lo primero que tenemos es el alfabeto propuesto, como el analizador está basado en Python se crearon varias clases dependiendo el tipo de token que se podrían encontrar como lo son las palabras reservadas, los identificadores, los operadores ya sean aritméticos, de comparación etc, y luego está la clase Token, que permitirá clasificar los tokens por tipo, lexema, línea y columna.

Paquete de Gramática:

Aquí se encuentran todas las clases que verifican la estructura del código, estas clases se programaron en base a ciertos autómatas mostrados en el documento de Gramática, estas clases reciben un arreglo de Tokens linea por linea para verificar si su estructura esté bien definida.



Paquete de Alfabeto:

En este paquete se encuentran las clases de tipo alfabeto en donde estas clases nada más cuentan con una lista de palabras pertenecientes al alfabeto definido, por ejemplo en el caso de la clase PalabrasClave, en este clase se pueden encontrar una lista de todas las palabras reservadas posibles de este analizador, además de que cada clase contiene un método que devuelve un booleano al recibir una palabra, este booleano podría ser verdadero si la palabra pertenece a la lista de palabras clave (en el contexto de la clase PalabrasClave) o falso si la palabra no pertenece a esta lista. Es la misma estructura aplicada en las demás clases del paquete de Alfabeto.

Paquete Token:

En el paquete Modelo, existe otro paquete llamado token, aquí podremos encontrar la clase Token. Esta clase es prácticamente un POJO, donde los atributos son el tipo, lexema, línea y columna, además de sus métodos getters y setters.

Paquete EstructuraSintactica:

Este es un paquete que se encuentra en el Modelo. prácticamente la clase EstructuraSintáctica contiene el

arreglo de Tokens, mensajes de error si es que la línea está mal estructurada, un booleano para definir si la línea está bien escrita

Controlador:

Aquí encontraremos únicamente la clases de AnalizadorLexico y AnalizadorSintactico

El **AnalizadorLexico** que se encarga de generar los tokens a partir del código enviado por el usuario, esta clase está compuesta por los atributos de la gramática y estados, además de una lista de tokens. En esta clase tendremos varios métodos que nos permitirán analizar y ordenar los tokens.

El método más importante de esta clase es **analizador(String codigoFuente)** que recorre el código fuente caracter por caracter utilizando un ciclo for, dentro de este for habrá un ciclo switch que tendrá la función de pasar a los estados correspondientes dependiendo del tipo de token que podría tratarse, como primer punto tenemos el ESTADO_INICIAL, en este estado se verificará el carácter y dependiendo de ciertos patrones de este carácter podría mandarse a los otros estados

```
case ESTADO_INICIAL:
    if (Character.isLetter(ch: caracter) || caracter == '_') {
        // Comenzar a leer un identificador
        lexema.append(c: caracter);
        if(Character.isLetter(ch: codigoFuente.charAt(i+1)) || codigoFuente.charAt(i+1) == '_'){
            estadoActual = ESTADO_LEYENDO_IDENTIFICADOR;
        }else{
            tokens.add(new Token(tipo: identificador.getNombreToken(), patron: "[a-zA-Z_][a-zA-Z0-9_]*", valo
            lexema.setLength(newLength: 0); // Reiniciar el lexema
            estadoActual = ESTADO_INICIAL; // Volver al estado inicial
        }

    } else if (caracter == '\n') {
        // Nueva linea, actualizar posición
        linea++;
        columna = 1;

    } else if (comentarios.verificandoPalabra(palabra:Character.toString(c: caracter))) {
        //SI LA LINEA EMPIEZA CON UN # ES UN COMENTARIO
        lexema.append(c: caracter);
        estadoActual = ESTADO_COMENTARIO;

    }else if (aritmeticos.verificandoPalabra(palabra:Character.toString(c: caracter)) && codigoFuente.charA
        //SI LA LINEA EMPIEZA CON UN ARITMETICO
        lexema.append(c: caracter);
        if(codigoFuente.charAt(i+1) == '*' || codigoFuente.charAt(i+1) == '/' ){
            estadoActual = ESTADO_ARITMETICO;
```

Luego de que el estado inicial cambie a los otros estados correspondientes, se llamarán ciertos métodos, que dependiendo a ciertas condiciones del carácter, se irán guardando hasta formar el lexema para llegar a una condición final y serán capturados por el objeto token y lo guardarán en un listado de este tipo de objetos, una vez recorrido todos los caracteres del código fuente, se devolverá un listado de

tokens que los manejará la vista para mostrarle los resultados al usuario

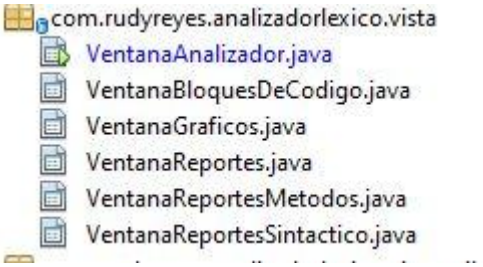
```
    break;
case ESTADO_LEYENDO_IDENTIFICADOR:

    leyendoIdentificador(caracter, i, codigoFuente);

    break;

case ESTADO_COMENTARIO:
    leyendoComentarios(caracter, i);

    break;

case ESTADO_ARITMETICO:

    leyendoAritmeticos(caracter, i);

    break;

case ESTADO_ASIGNACION:
    leyendoAsignacion(caracter, i);

    break;
case ESTADO_COMPARACION:
    leyendoComparacion(caracter, i);
```

En la clase **AnalizadorSintactico**, prácticamente se obtienen los tokens línea por línea y son enviados a las clases de Gramática para verificar su estructura estas devuelven un objeto de tipo EstructuraSintactica en donde se puede verificar si la línea está bien estructurada.

```
public static List<EstructuraSintactica> analizarSintaxis (List<Token> tokens ){
    List<EstructuraSintactica> estructuraSintactica = new ArrayList<>();
    int filaMaxima = filaMaxima(tokens);

    for (int fila = 1; fila <= filaMaxima; fila++) {
        List<Token> tokensFilaActual = new ArrayList<>();

        // Obtener los tokens para la fila actual
        for (Token token : tokens) {
            if (token.getLinea()== fila) {
                tokensFilaActual.add(e: token);
            }
        }

        // Llama a tu método y pasa los tokens de la fila actual
        if(!tokensFilaActual.isEmpty()){

            //LLAMAR PRINT
            if(tokensFilaActual.get(index: 0).getValor().equals(anObject: "print")){
                estructuraSintactica.add(e: funcionPrint.analizarPrint(tokens: tokensFilaAct
```

Vista:

Aquí podremos encontrar varias clases. estas clases son las encargadas de pedir al usuario que escriba el código y conectarlas al controlador, ademas hay varias clases que nos permitirán ver los reportes generados.



VentanaAnalizador:

Esta ventana será la principal, se encargará de recibir el archivo de entrada que el usuario podría mandar, además de ser la parte en donde el usuario ingrese el código, lo edite y lo elimine, además de generar los tokens, también cuenta con botones que permitirán desplegar las otras dos ventanas.

```
private void subirArchivoActionPerformed(java.awt.event.ActionEvent evt) {

private void ejecutarCodigoActionPerformed(java.awt.event.ActionEvent evt)

private void resaltarTexto(){...83 lines }


private void generarGraficoActionPerformed(java.awt.event.ActionEvent evt)

private void reportesBotonActionPerformed(java.awt.event.ActionEvent evt) {
```

Cuenta con varios métodos, cada método corresponde a una acción generada por los botones que existen en la ventana.

En el método subirArchivo, el usuario selecciona el archivo a subir y este será cargado en el área de texto, en el método ejecutarCodigo prácticamente se obtendrá el codigo del area de texto y será enviado a la clase AnalizadorLexico para obtener los tokens generados, en los últimos dos métodos, nos permitirán abrir las otras dos ventanas, ya sean la de reportes y gráficos.

VentanaReporte:

```
private List<Token> tokens;
public VentanaReportes(java.awt.Frame parent, boolean modal,  List<Token> tokens) {
    super(owner: parent, modal);
    this.tokens= tokens;
    initComponents();

    DefaultTableModel model = (DefaultTableModel) tablaReportes.getModel();
    for (Token token : tokens) {
        if (!token.getTipo().equals(anObject: "Error")) {
            model.addRow(new Object[]{
                token.getTipo(),
                token.getPatron(),
                token.getValor(),
                token.getLinea(),
                token.getColumna()
            });
        }
    }
}
```

Esta ventana se encarga de recibir todos los tokens generados por la clase AnalizadorLexico, y organizarlos en una tabla, por tipo, patrón, lexema, fila y columna . para despues desplegarlas en la ventana de reportes y que el usuario pueda mirarlos

VentanaGraficos:

Esta ventana desplegará todos los tokens disponibles para graficar, haciendo uso de la librería Graphviz, a esta ventana serán enviados los tokens, y los ordenará de modo que queden en una lista desplegable, luego cuando el usuario le de click a un lexema en especifico se mostrará una gráfica del lexema seleccionado

VentanaReportesSintacticos:

Esta clase se encargará de recibir las líneas de códigos procesadas por el AnalizadorSintactico y mostrar la tabla de símbolos global, además de tener otras opciones como la tabla de símbolos por bloque de código o ver los métodos definidos.

VentanaBloquesDeCodigo:

Aqui se mostrarán todos los bloques de código encontrados además de mostrar la tabla de símbolos de cada bloque.

VentanaReportesMetodos:

Esta clase servirá para mostrar todos los métodos definidos en el código, también mostrará los métodos que fueron invocados en el código.

3. Instalacion y Configuracion:

Para usar este programa solamente se necesita tener JAVA instalado, el programa fue desarrollado utilizando JAVA 17, así que es probable que no funcione en versiones anteriores de JAVA 8, debido a que el proyecto dispone de un archivo ejecutable no es necesario tener un IDE como lo es Apache Netbeans para ejecutarlo, desde una terminal podría funcionar sin problemas. si se ejecuta desde un IDE probablemente se necesiten descargar algunas librerías como lo es Graphviz, que nos permitirán graficar los lexemas generados.