Nueva Guatemala de la Asunción

Manual de Técnico

Proyecto 1: OLC-1

Elaborado por: Estuardo Gabriel Son Mux

Carné: 202003894

Fecha: 23/02/2022

Índice

[Alcances y Objetivos del Programa 3](#_Toc96714259)

[Especificaciones Técnicas 4](#_Toc96714260)

[Requisitos de Hardware 4](#_Toc96714261)

[Requisitos de Software 4](#_Toc96714262)

[Sistema Operativo 4](#_Toc96714263)

[Lenguaje de Programación 4](#_Toc96714264)

[IDE o Editor de Código 4](#_Toc96714265)

[Lógica del Programa 5](#_Toc96714266)

[Estructura de Archivos EXP 5](#_Toc96714267)

[Archivos Utilizados 6](#_Toc96714268)

[Package Analizadores 6](#_Toc96714269)

[Package Arboles 9](#_Toc96714270)

[Package ocl1\_proyecto1 25](#_Toc96714271)

# Alcances y Objetivos del Programa

El programa realizado tiene como objetivo el análisis léxico y sintáctico de archivos EXP (extensión “.exp”), los cuales tienen la información para la creación de árboles y autómatas tanto deterministas con el método del árbol, como no determinista con el método de Thompson y graficarlos por medio de la librería Graphviz o tablas de HTML. Posteriormente al análisis se procede a realizar un análisis de las frases incluidas en el archivo EXP con una de las expresiones para verificar su validez.

# Especificaciones Técnicas

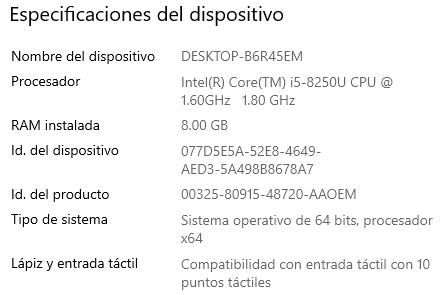
## Requisitos de Hardware

* Computadora con todos sus componentes para su correcto funcionamiento
* Laptop

## Requisitos de Software

### Sistema Operativo

El programa fue desarrollado en una laptop con sistema operativo Windows 10, Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz.



### Lenguaje de Programación

JAVA

#### Librerías

* Html2image
* Java-cup-11a
* Java-cup-11b
* JFlex
* Json-simple

### IDE o Editor de Código

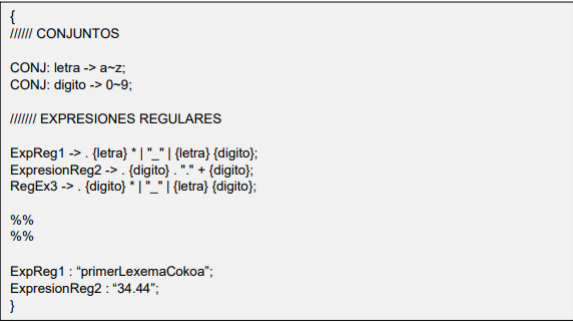
Apache Netbeans IDE 12.6



# Lógica del Programa

## Estructura de Archivos EXP

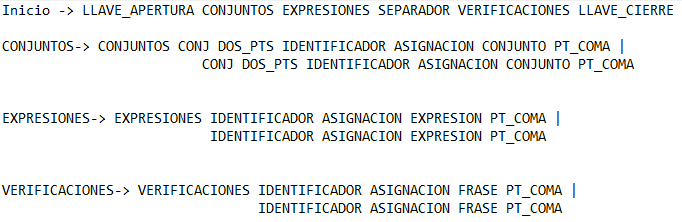
Los archivos EXP poseen la siguiente estructura.



Para ellos se hizo un reconocimiento de tokens con la siguiente tabla de tokens.

|  |  |  |
| --- | --- | --- |
| Token | Lexema | Patrón |
| LLAVE\_APERTURA | { | { |
| LLAVE\_CIERRE | } | } |
| DOS\_PTS | : | : |
| PT\_COMA | ; | ; |
| ASIGNACION | -> | -> |
| SEPARADOR | %% | (\%)+ |
| CONJ | CONJ | CONJ |
| IDENTIFICADOR | Expresion1 | (\_)\*[a-zA-ZnÑ]+[\_a-zA-Z0-9ñÑ]\* |
| CONJUNTO | a,e,i,o,u | ([a-z]\~[a-z])|([^a-zA-Z0-9\~]\~[^a-zA-Z0-9\~])|([A-Z]\~[A-Z])|([0-9]\~[0-9])|([^\~](,[^\~])+) |
| EXPRESION | ..+{digito}"."+{digito} | [\.\|\\*\+\?]([a-zA-Z0-9ñÑ\|\?\+\\*\.]|(\"((\\[^\n])|([^\\\"\n]))+\")|(\{(\_)\*[a-zA-ZnÑ]+[\_a-zA-Z0-9ñÑ]\*\}))+ |
| ESPACIO |  | [\ \r\t\f\t] |
| SALTO | \n | [\ \n] |
| FRASE | "34.51" | \”(\\\”|\\n|\\\’|[^\"])\” |

Para el análisis sintáctico se utilizó la siguiente gramática:

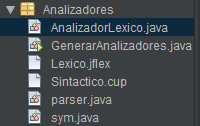


## Archivos Utilizados

Para la elaboración del proyecto en Java se separaron los archivos en 3 paquetes los cuales son:

* **Analizadores:** En este paquete se almacenan los archivos utilizados para la creación de los analizadores, tanto léxico como sintáctico.
* **Arboles:** En este paquete se almacenan todos los archivos utilizados para la creación de las estructuras para la realización del método del árbol y los autómatas finitos deterministas y no deterministas.
* **Ocl1\_proyecto1:** En este paquete se encuentran los archivos para la creación de la ventana en donde se visualizan todos los componentes y la ejecución de las funciones.

### Package Analizadores



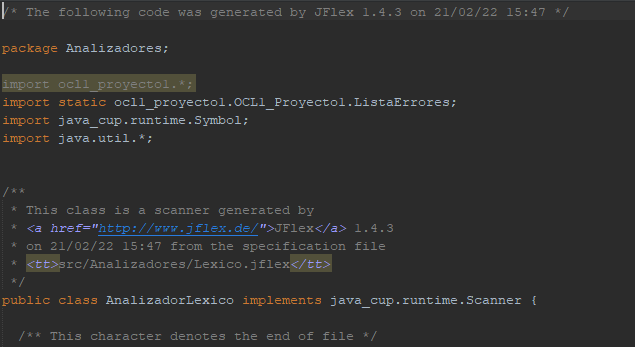
#### Analizadores.java

En este archivo se encuentra la generación de los archivos AnalizadorLexico.java, parser.java y sym.java, esto mediante la utilización de las librerías JFlex y Cup que mediante la utilización de archivos de extensión .jflex y .cup genera dichos archivos para el análisis Léxico y Sintáctico respectivamente.

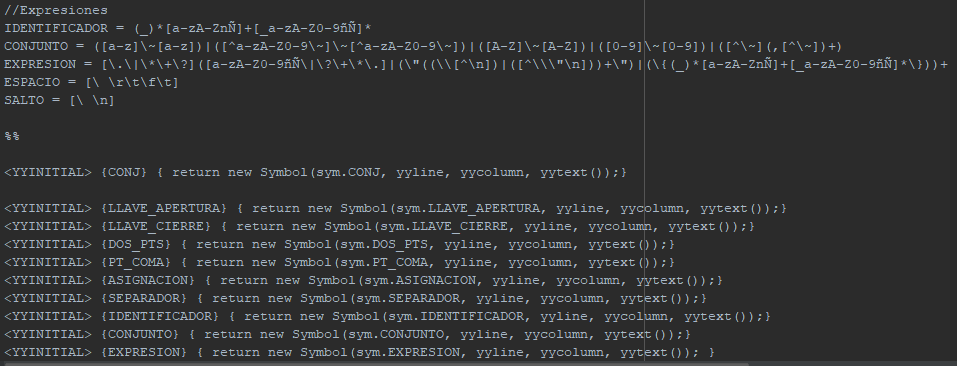
#### 

#### AnalizadorLexico.java

Archivo generado por medio de la librería JFlex para el análisis sintáctico de los archivos de entrada del programa.



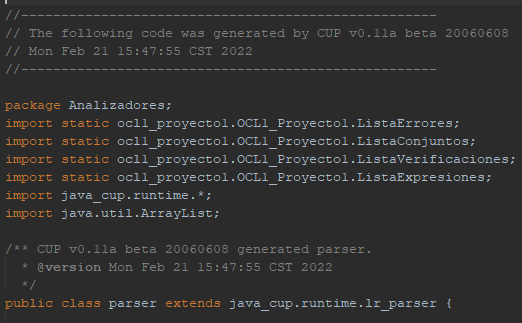
*Parte del código de AnalizadorLexico.java.*



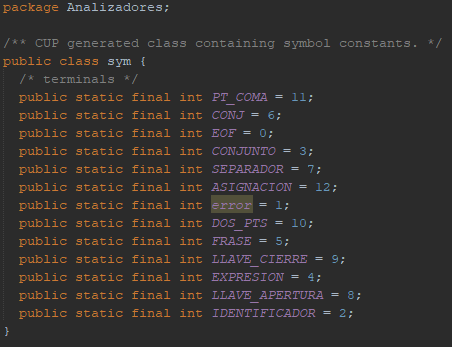
*Parte de la sintaxis del archivo Lexico.jflex.*

#### Parser.java y sym.java

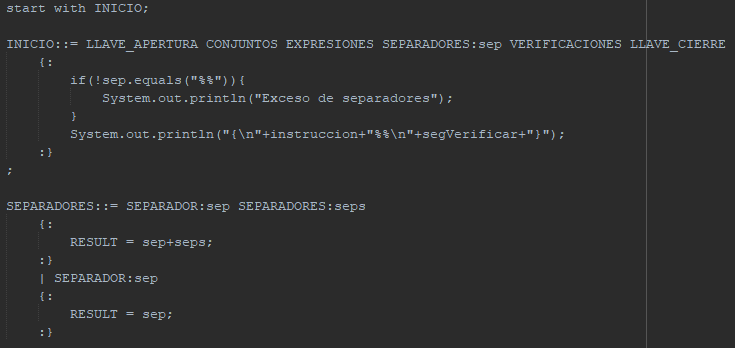
Son archivos generados por medio de la librería java-cup mediante el uso del archivo Sintactico.cup los cuales son de utilidad para el análisis Sintáctico del archivo de entrada durante la ejecución del programa. Esto permite la recolección de errores, así como los conjuntos, expresiones y frases dentro del archivo EXP.



*Parte del código de parser.java.*

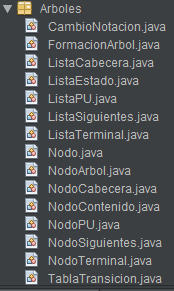


*Parte del código de sym.java.*



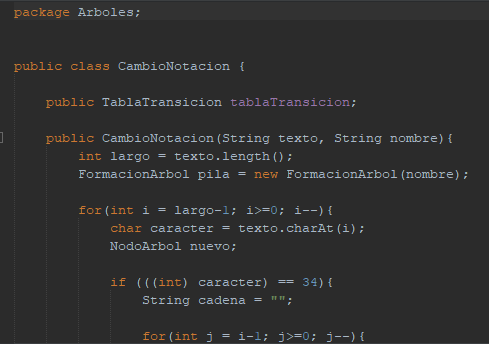
*Parte del contenido del archivo Sintactico.cup.*

### Package Arboles

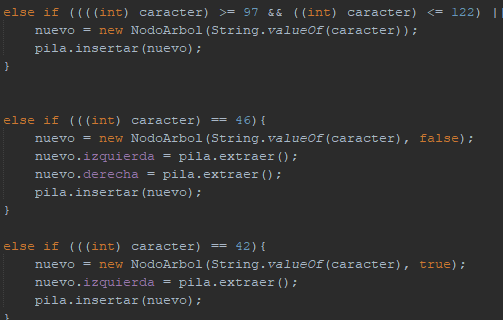


#### CambioNotacion.java

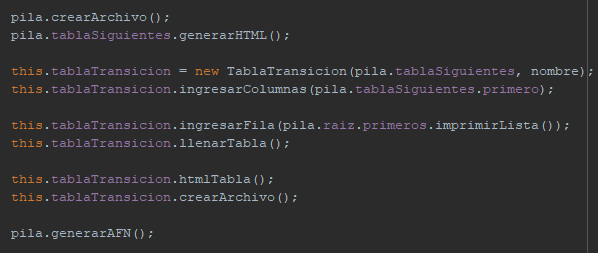
En este se utiliza para como centro para el llamado a las funciones y procedimiento para la realización del método del árbol, la tabla de siguientes y tabla de transiciones para la formación del autómata finido determinista y el método de Thompson para la formación del autómata finito no determinista. Esto lo hace recibiendo como dos parámetros los cuales son la expresión regular la cual se desea realizar el árbol y el nombre que recibe dicha expresión.



Posteriormente realiza una lectura de la cadena carácter por carácter de derecha a izquierda realizando validaciones para ingresarlos como NodoArbol dentro de una pila y extrayendo la cantidad necesaria para la concatenación o cualquier otra operación que indique la expresión.

  
*Ejemplos de validaciones para la formación del árbol.*

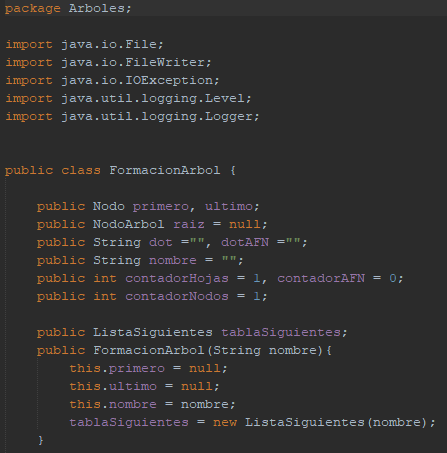
Por último, se ejecutan los métodos de la pila con la raíz del árbol para la creación de las tablas de siguientes, transiciones y la gráfica de los autómatas.



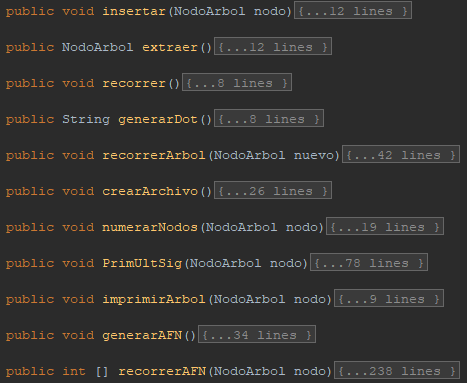
*Implementación de las funciones de pila para la creación de los reportes.*

#### FormacionArbol.java

En este archivo se encuentran todos los procedimiento y funciones para la creación de una pila de NodoArbol, mediante la implementación de la clase nodo que implementara la relación entre la pila.

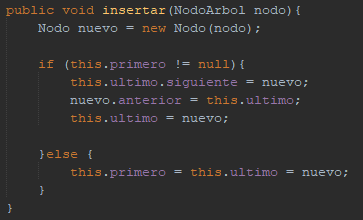


##### Métodos en FormacionArbol.java



###### Método insertar(NodoArbol nodo)

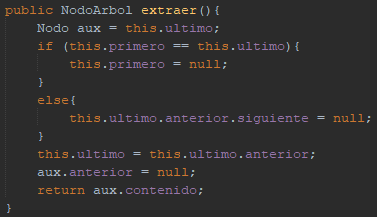
Método que se utiliza para ingresar nodos del tipo NodoArbol dentro de la pila.



*Código del método insertar.*

###### Función extraer()

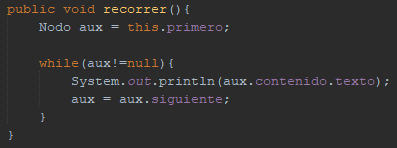
Función que devuelve el contenido del primer nodo de la pila.



*Código de la función extraer.*

###### Método recorrer()

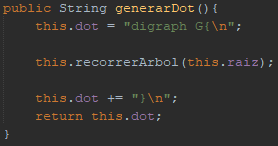
Método para imprimir en consola el contenido de la pila.



*Código del método recorrer.*

###### Función generarDot()

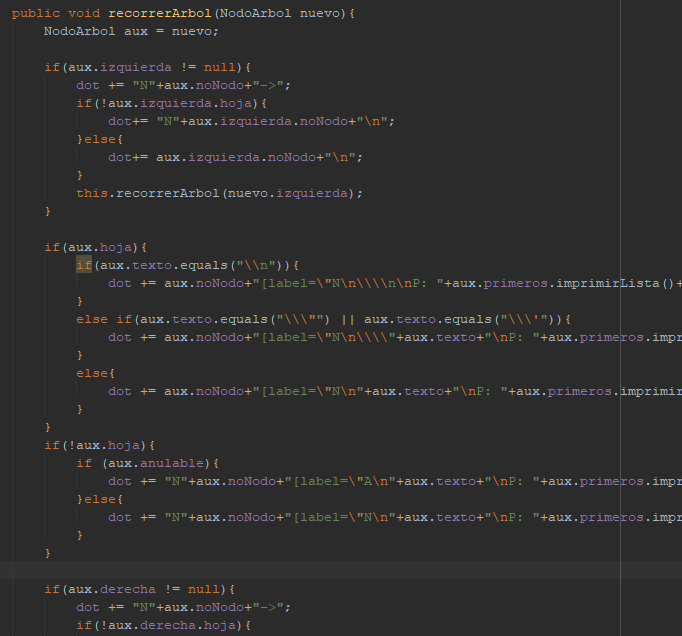
Función que se utiliza para generar el archivo dot que contiene la información para generar el árbol del método del árbol, retornado la información como una cadena String.

**

*Código de la función generarDot.*

###### Método recorrerArbol(NodoArbol nuevo)

Método utilizado para recorrer el árbol para concatenar a la variable String dot la información de cada nodo perteneciente al árbol.



*Parte del código del método recorrerArbol.*

###### Método crearArchivo()

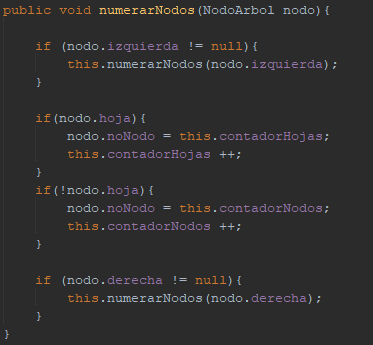
Método para la creación del archivo .dot y la creación de la imagen .png dentro de la capeta Arboles\_202003894.



*Método crearArchivo.*

###### Método numerarNodos(NodoArbol nodo)

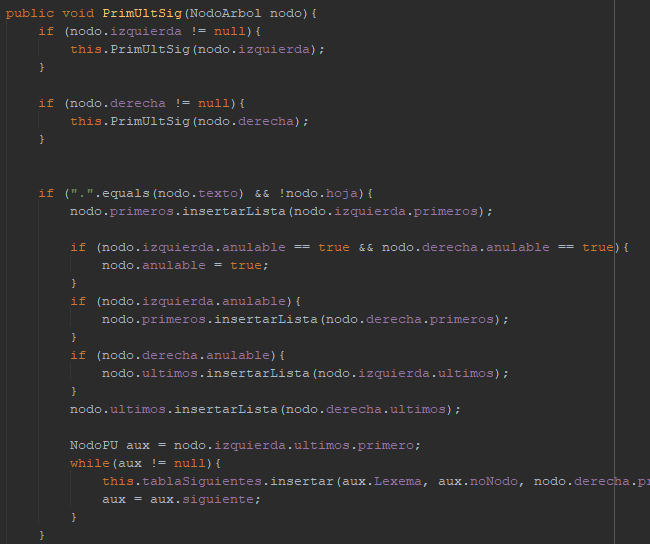
Método para recorrer el árbol e ir numerando los nodos hijos del árbol en orden ascendente de forma recursiva.



*Método numerarNodos.*

###### Método PrimUltSig(NodoArbol nodo)

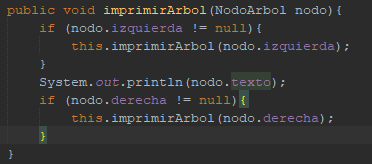
Método para modificar los atributos del cada uno de los NodoArbol pertenecientes al método del árbol para en ellos el listado de nodos primeros, últimos y siguientes.



*Parte del código del método PrimUltSig.*

###### Método imprimirArbol

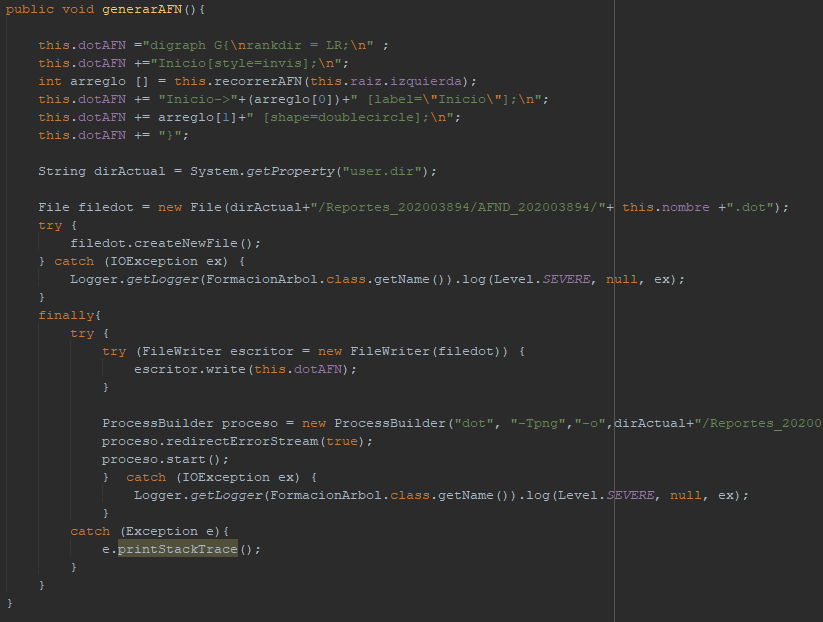
Método para imprimir en consola el árbol generado en in-order.



*Código del método*

###### Método generarAFN()

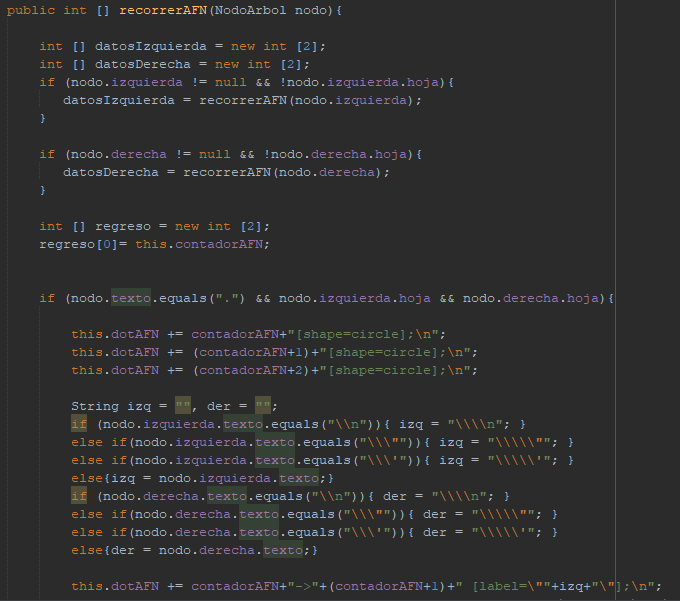
Método para la creación del archivo .dot y .png del método de Thompson.



*Código del método generarAFN.*

###### Función recorrerAFN(NodoArbol nodo)

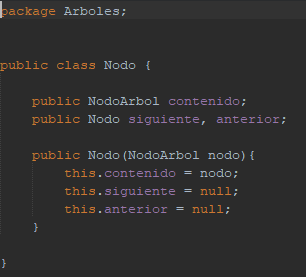
Función recursiva que recorre todos los nodos del árbol del método del árbol para generar el método de Thompson y generar así un String con formato DOT con el cuál generar la gráfica del mismo.



*Parte del código de la función recorrerAFN.*

#### Nodo.java

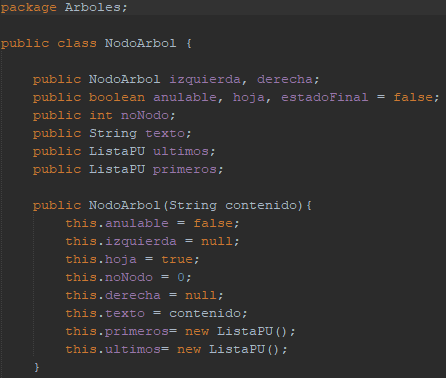
Archivo utilizado para la creación de los objetos nodos que formaran la pila.

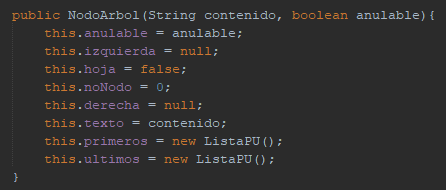


*Código de Nodo.java.*

#### NodoArbol.java

Archivo utilizado para la creación de los nodos que formaran parte del árbol correspondiente a cada uno de los arboles generados por el método del árbol.

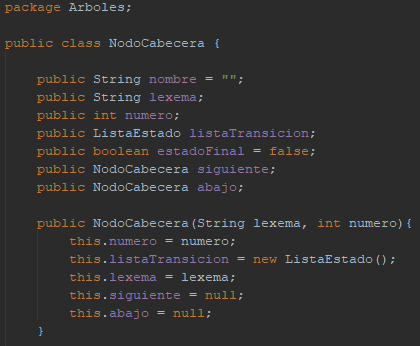


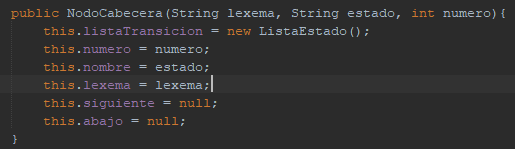


*Código de NodoArbol.java*

#### NodoCabecera.java

Con esta clase se crean los objetos que servirán como cabecera de las filas y columnas de la tabla de transición. Posee dos constructores dependiendo del caso en que se desee crear un nodo cabecero para fila o columna.

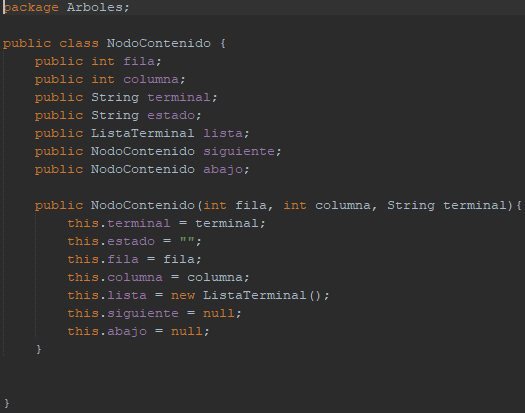




*Código de Nodocabecera.java*

#### NodoContenido.java

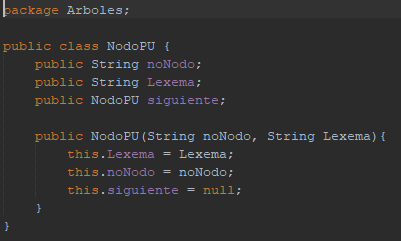
Con esta clase se crean los objetos Nodo que irán dentro y formaran la tabla de transiciones.



*Código de NodoContenido.java*

#### NodoPU.java

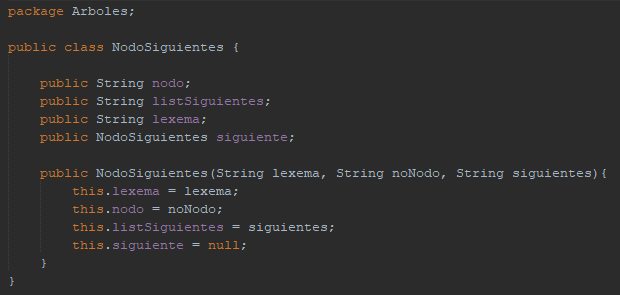
Archivo que contiene la clase para la creación de los nodos que formarán parte de la lista de Nodos que serán los primero o últimos de un nodo NodoArbol.



*Código de NodoPU.java*

#### NodoSiguientes.java

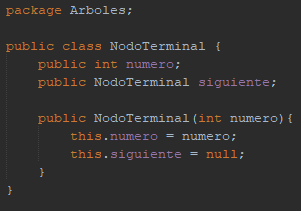
Archivo que contiene la clase para la creación de los nodos que formarán parte de la lista de Nodos siguientes de un terminal.



*Código de NodoSiguientes.java*

#### NodoTerminal.java

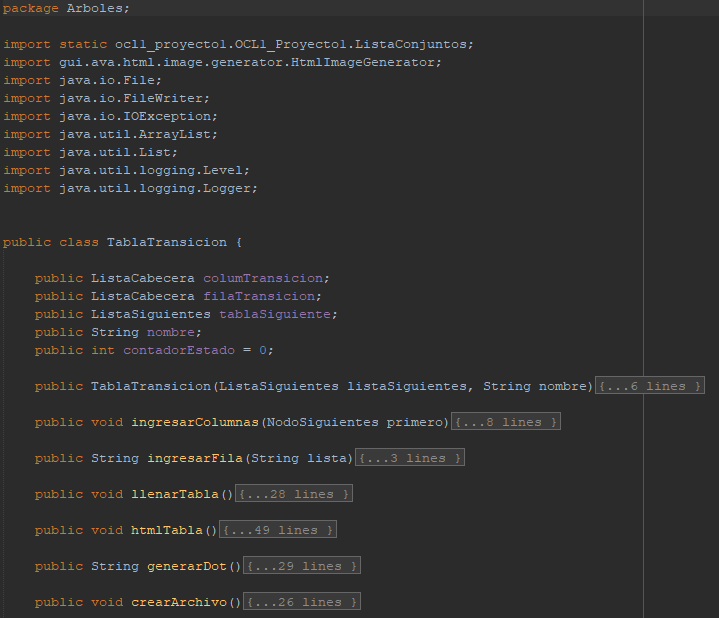
Archivo que contiene la clase para la creación de los nodos que formaran parte de la ListaTerminal la cual contiene el numero de terminal que pertenece a un estado.



*Código de NodoTerminal.java*

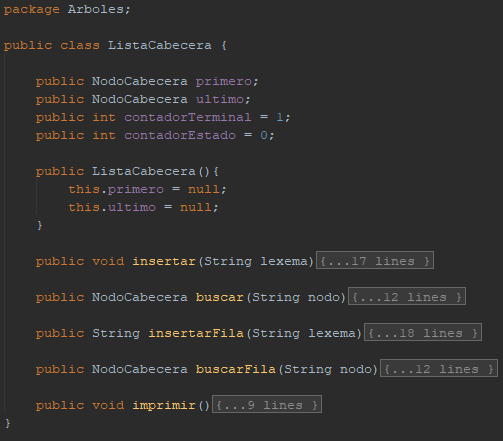
#### TablaTransiciones.java

Archivo que contiene la clase para la creación de la tabla de transiciones la cual funciona como una matriz ya que contiene dos listas de nodos cabeceras correspondientes a los estados (filas) y terminales (columnas).



#### ListaCabecera.java

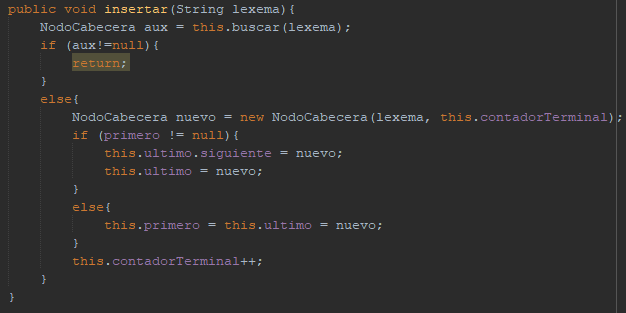
Clase utilizada para la creación de la lista de nodoCabecera que forman la tabla de transiciones.



##### Métodos y funciones de ListaCabecera.java

###### Método Insertar(String Lexema)

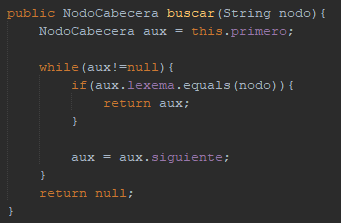
Método para insertar nodos dentro de la Lista de cabeceras para formar las columnas.



*Código del método Insertar.*

###### Función buscar(String nodo)

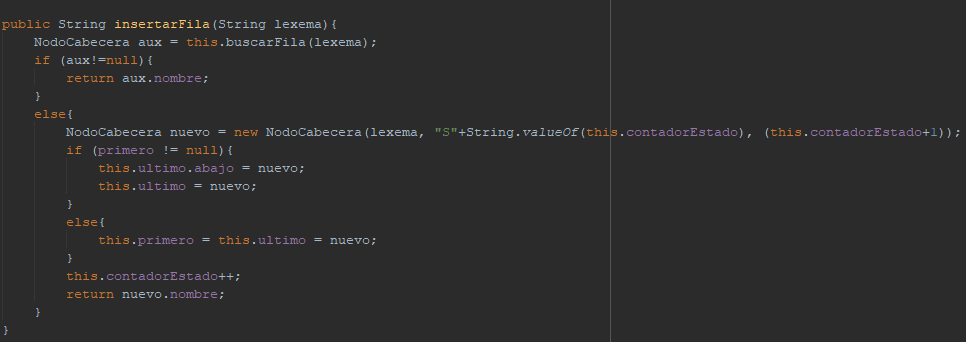
Función que retorna un objeto del tipo NodoCabecera si encuentra una coincidencia con la cadena de texto que recibe como parámetro en una lista que sea de columnas.



*Código de la función buscar.*

###### Función InsertarFila(String Lexema)

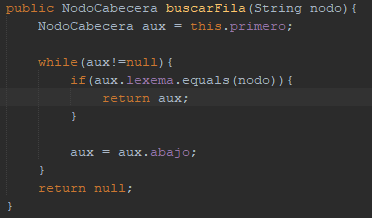
Función para insertar nodos dentro de la Lista de cabeceras para formar las Filas y retorna una cadena String con el nombre del Estado creado.



*Código del método Insertar.*

###### Función buscarFila(String nodo)

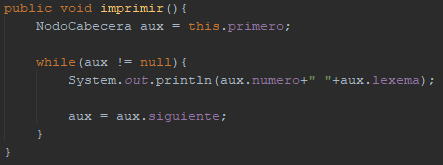
Función que retorna un objeto del tipo NodoCabecera si encuentra una coincidencia con la cadena de texto que recibe como parámetro en una lista que sea de Filas.



*Código de la función buscarFila.*

###### Método imprimir()

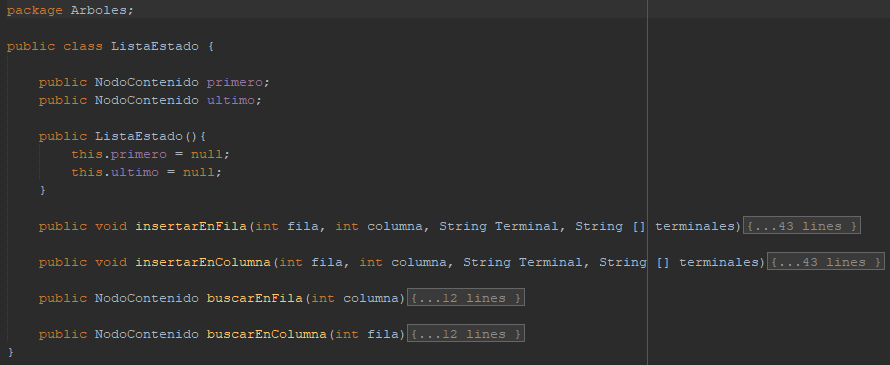
Método para imprimir en consola la lista de columnas, correspondiente a los terminales.



*Código del método imprimir.*

#### ListaEstado.java

Archivo que sirve para la creación de la lista de Estados que puede tener un NodoCabecera del tipo Fila.

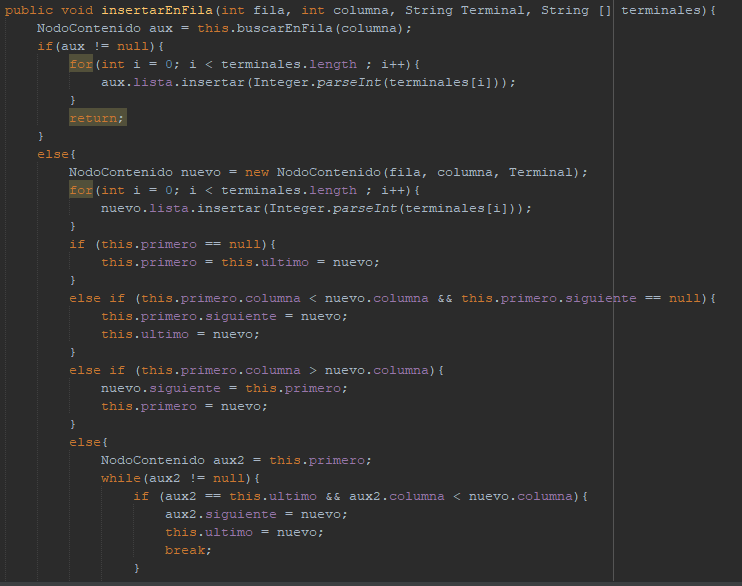


*Código de Listado.java*

##### Métodos y funciones de ListaEstado.java

###### Método insertarEnFila(int fila, int columna, String Terminal, String [] terminales)

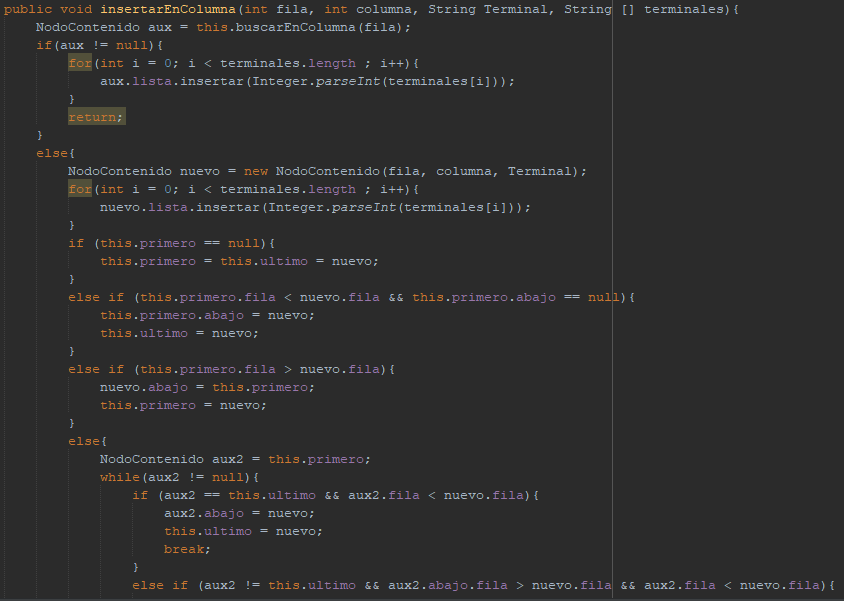
Método que inserta en la lista de forma ascendente cada nuevo nodo a ingresar.



*Parte del código de la función insertarEnFila.*

###### Método insertarEnColumna(int fila, int columna, String Terminal, String [] terminales)

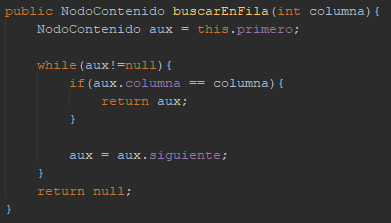
Método que inserta en la lista de forma ascendente cada nuevo nodo a ingresar.



*Parte del código de la función insertarEnColumna.*

###### Función buscarEnFila(int columna)

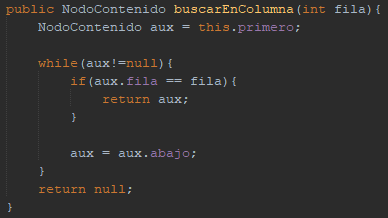
Función que retorna un objeto del tipo NodoContenido si encuentra una coincidencia con el número de columna que recibe como parámetro en la lista actual.



*Código de buscarEnFila.*

###### Función buscarEnColumna(int fila)

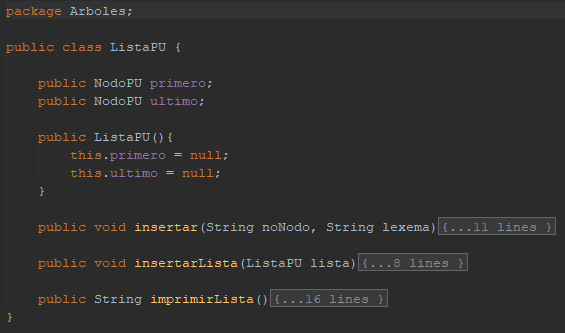
Función que retorna un objeto del tipo NodoContenido si encuentra una coincidencia con el número de fila que recibe como parámetro en la lista actual.



*Código de buscarEnColumna.*

#### ListaPU.java

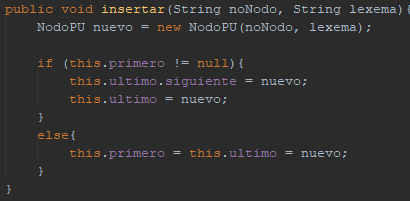
En este archivo se encuentra la clase utilizada para la creación d ellos listados de nodos Primeros y Últimos de un nodoArbol.



##### Métodos y funciones de ListaPU.java

###### Método insertar(String noNodo, String Lexema)

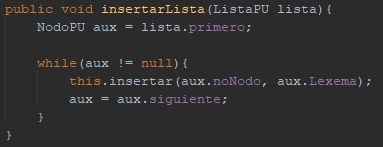
Método que inserta dentro de la Lista de primeros y últimos un nuevo nodo.



*Código método insertar.*

###### Método insertarLista(ListaPU lista)

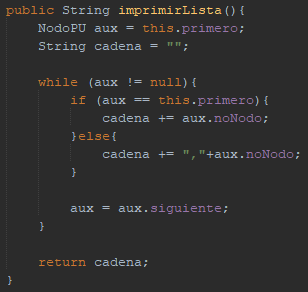
Método para insertar varios nodos dentro de la lista Primeros o últimos sucesivamente.



*Código de insertarLista.*

###### Función imprimirLista()

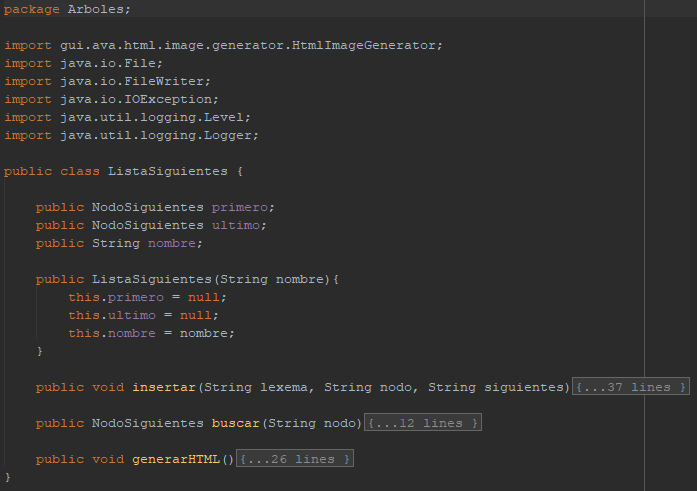
Función que retorna una cadena de texto con el contenido de toda la lista actual.



*Código de imprimirLista.*

#### ListaSiguientes.java

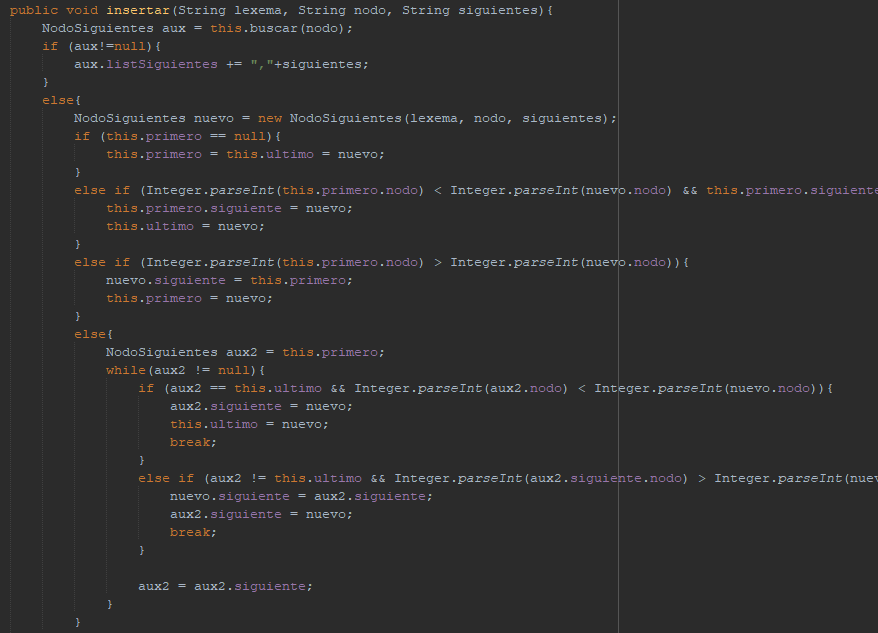
En este archivo se encuentra la clase con la cual se crean las listas de nodos siguientes a un nodo terminal.



##### Métodos y funciones de ListaSiguientes.java

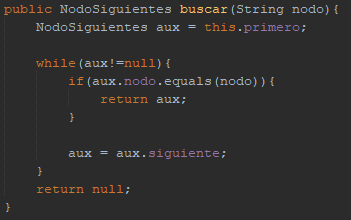
###### Método insertar(String lexema, String nodo, String siguientes)

Método que ingresa en orden ascendente cada nodo que se desee ingresar dentro de la lista.



###### Función buscar(String nodo)

Función que retorna un objeto del tipo NodoSiguientes el cual su atributo nodo sea igual al parámetro nodo que recibe la función.



*Código de buscar.*

###### Método generarHTML()

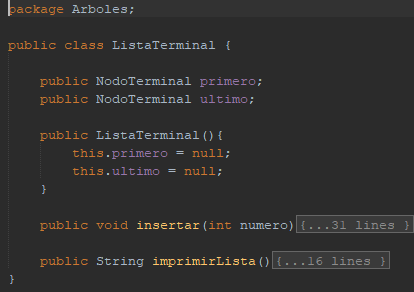
Método que genera un archivo html dentro de la carpeta Siguientes\_202003894 y una imagen png con el contenido de la tabla html.



*Código de generarHTML.*

#### ListaTerminal.java

Archivo que contiene la clase para la creación de Listas de Terminales que posee cada estado de la tabla de transiciones.

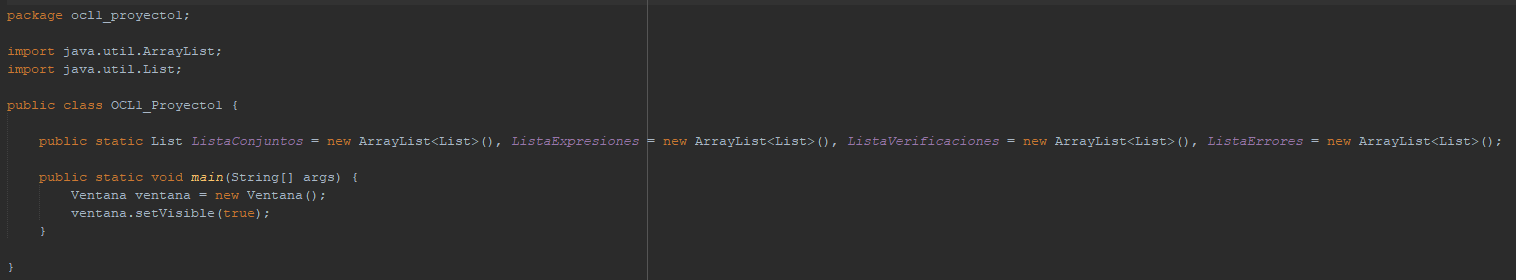


### Package ocl1\_proyecto1



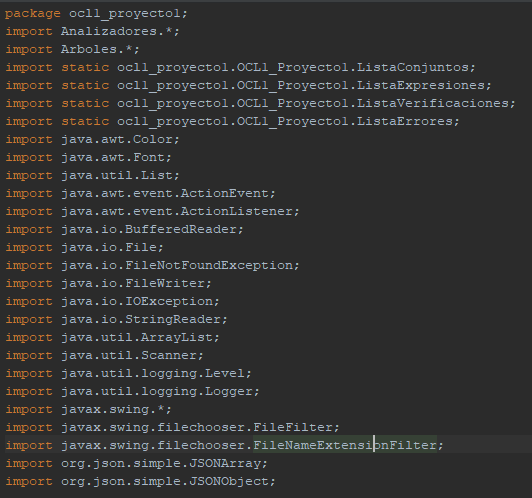
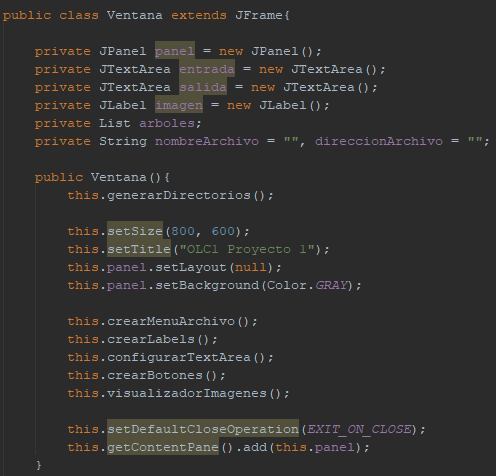
#### OCL1\_Proyecto1.java

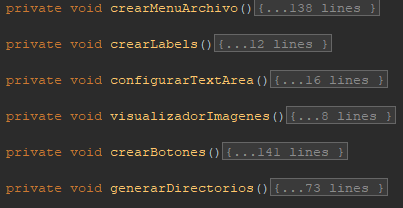
Archivo principal del proyecto que contiene la inicialización de las listas: ListaConjuntos, ListaExpresiones y ListaErrores. Así mismo tiene la inicialización de la ventana en que se mostrará la ejecución.



#### Ventana.java

Archivo que contiene la configuración de la ventana que se visualizara durante la ejecución de la ventana, esto incluye: apariencia y funcionamiento.

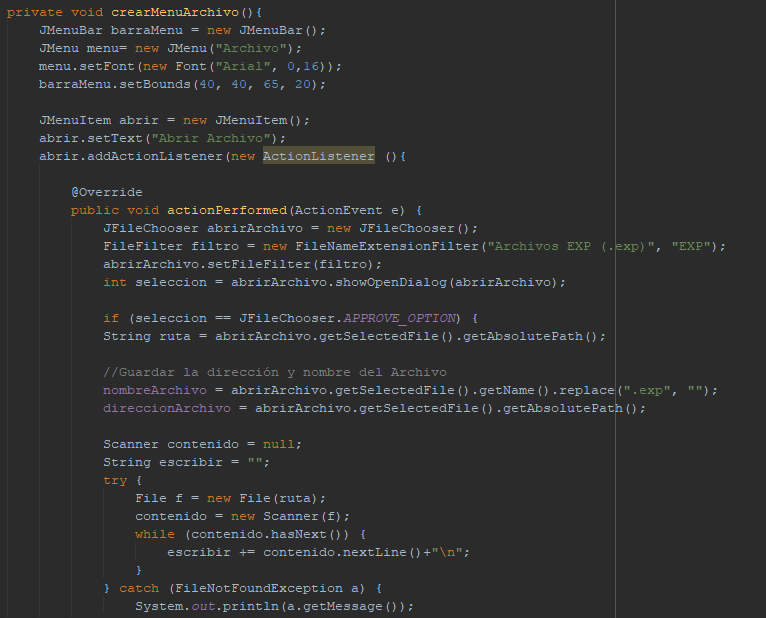
 



##### Métodos y funciones de Ventana.js

###### Método crearMenuArchivo()

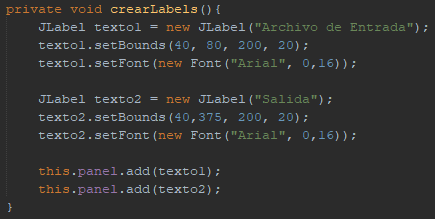
Este método se utiliza para crear el menú desplegable que contiene todas las opciones para la creación o apertura de archivos.



*Parte del código de crearMenuArchvo.*

###### Método crearLabels()

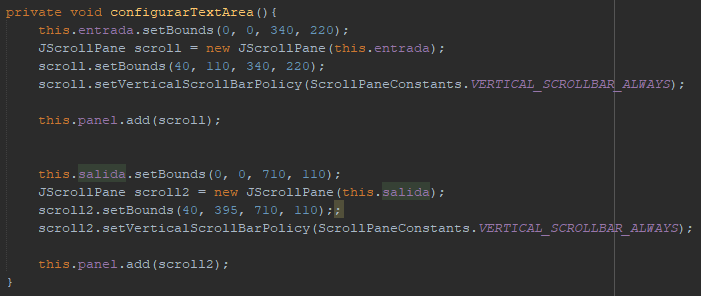
Método para crear los labels que contendrán texto para indicar que sección del programa es cada una.



*Código de crearLabels.*

###### Método configurarTextArea()

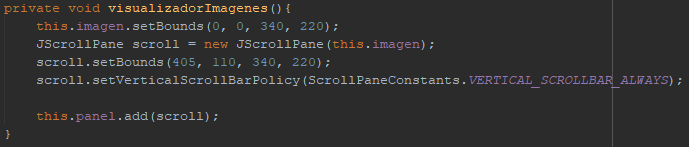
Método para crear las áreas de texto de entrada y salida de información.



*Código de configurarTextArea.*

###### Método visualizadorImagenes()

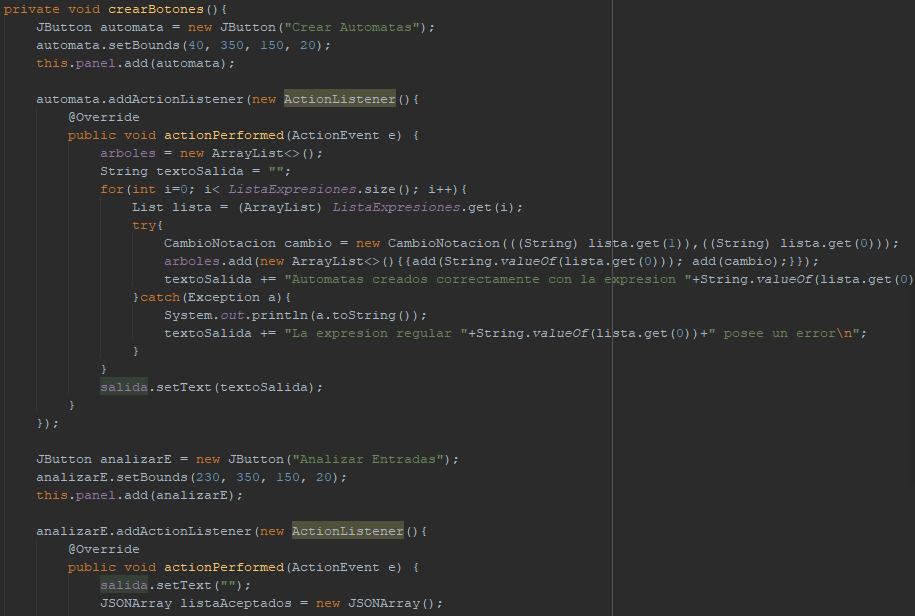
Método para crear el JScrollPane que se utiliza para mostrar las imágenes de reportes generados durante la ejecución.



*Código de visualizadorImagenes.*

###### Método crearBotones()

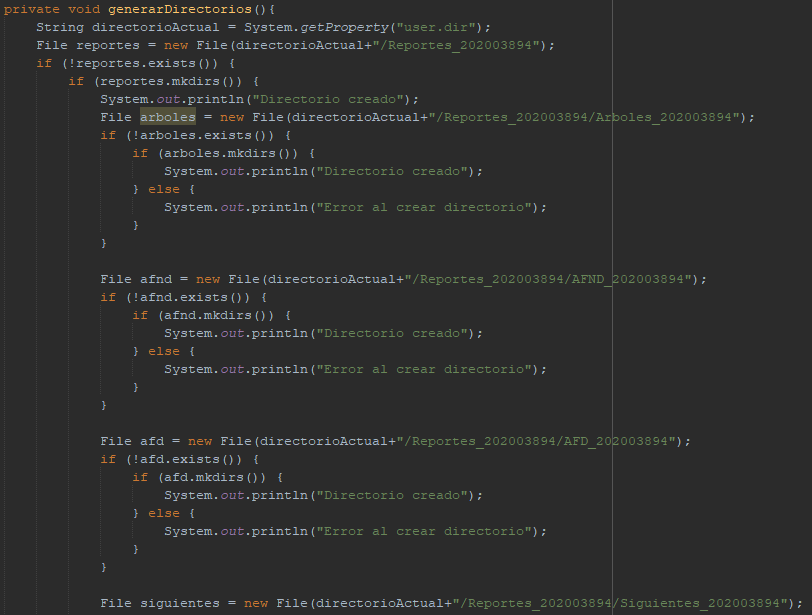
Método para configurar los botones con los que se ejecutaran las acciones del programa las cuales son: Analizar el archivo de entrada, Crear Autómatas, Analizar las Frases y Visualizar Imágenes.



*Parte del código de crearBotones.*

###### Método generarDirectorios()

Método para generar las carpetas donde se almacenarán los reportes que se generen durante la ejecución del programa.



*Parte del código de generarDirectorios.*