MANUAL TECNICO

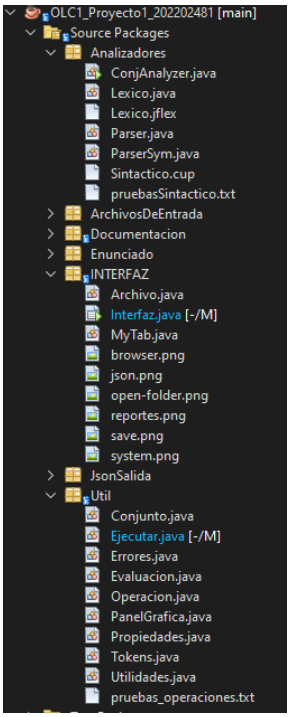
Nombre: Josué Nabí Hurtarte Pinto ---------- Carné: 202202481

El programa se desarrolló utilizando el lenguaje de programación Java y las herramientas JFLEX y CUP para la generación de los analizadores léxicos y sintácticos. Estos analizadores son fundamentales para la interpretación de las expresiones que se utilizan a la hora de Definir, Operar y Evaluar conjuntos. La estructura del programa se dividió en módulos claramente definidos para cada funcionalidad, incluyendo la entrada de datos, el análisis léxico y sintáctico, la realización de operaciones, la realización de simplificaciones a partir de las propiedades de los conjuntos y también graficar las operaciones. Además, se implementaron mecanismos para el manejo de errores léxicos y sintácticos, garantizando así la robustez y fiabilidad del sistema. La interfaz de usuario se diseñará de forma intuitiva y amigable, permitiendo al usuario ingresar los datos de entrada de manera sencilla, visualizar los resultados de las operaciones realizadas y generar los gráficos correspondientes de forma rápida y eficiente.

Requerimientos mínimos del entorno de desarrollo

- Java JDK21
- IDE (Entorno de Desarrollo Integrado) en nuestro caso Apache Neatbens / VS Code.
- Git
- Librería Jflex
- Librería Cup
- Librería JsonObject

Estructura del proyecto

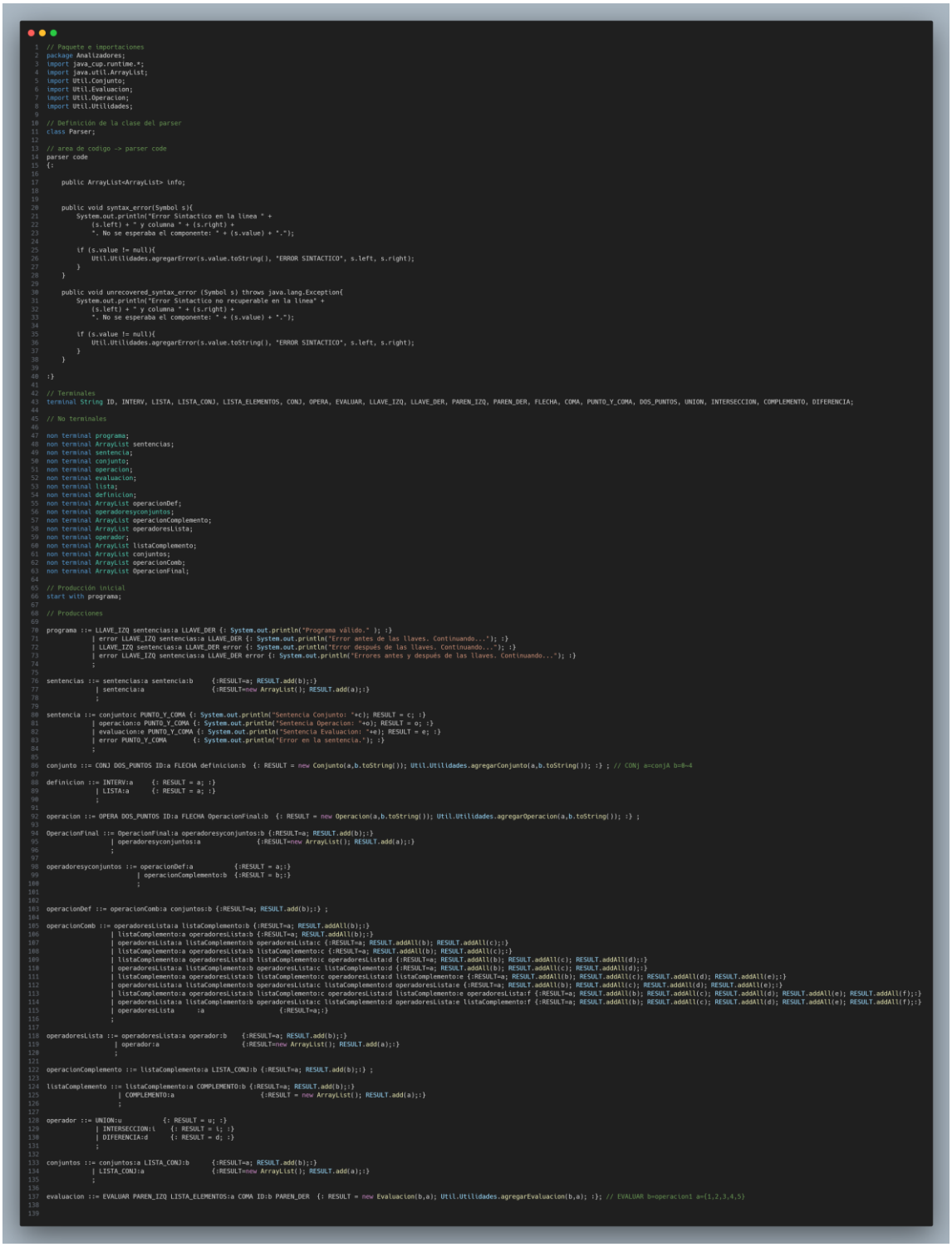


Analizadores: En este paquete se implementó los analizadores que incluye el analizador Léxico y analizador Sintáctico. Cada uno de estos analizadores desempeña un papel fundamental en la interpretación y comprensión del código fuente ingresado por el usuario.

Análisis Léxico: Para este analizador se implementó por medio de la librería Jflex que nos ayuda a realizar dicha acción por la cual se tuvo que definir expresiones regulares que nos ayuden a poder generar cada palabra reservada. El listado de dichas expresiones regulares, tokens, palabras reservadas:

```
//Paquete e importaciones
package Analizadores;

import java_cup.runtime.Symbol;
import java.util.ArrayList;
import java.util.LinkedList;
import Util.Utilidades;

%%
//Directivas

%class Lexico          // Nombre de la clase que genera JFlex
%public                // Para tener acceso desde otros paquetes
%line                  // Para registrar las líneas
%column                // Para registrar las columnas
%char                  // Llevar un conteo de caracteres
%cup                   // Habilita la integración con Cup
%unicode               // Reconocimiento de caracteres unicode
//%ignorecase

%init{                 // Constructor del analizador
    yyline = 1;
    yycolumn = 1;
%init}

// Expresiones regulares

BLANCOS = [ \t \r \n]+
COMENTARIO_LINEA = "#" .*
COMENTARIO_MULTILINEA = "<!"[^!]*"!>"

DIGITO = [0-9]
LETRA = [a-zA-ZÑñ]
ID = {LETRA}({LETRA}|{DIGITO}|"_")*
CARACTER = ([\x21-\x7E]) // Caracteres ASCII imprimibles (33-126) o (!-~)
INTERV = {CARACTER}("~"|" ~ "|"  ~  "){CARACTER} // a~z
LISTA = {CARACTER}(({BLANCOS})*","({BLANCOS})*{CARACTER})* // 1,2,3,4,5
LISTA_CONJ = ("{"{ID}"}") // {conjuntoA}
LISTA_ELEMENTOS = "{"({LISTA}|{INTERV})"}" // {1,2,3,4,5} {a,b,c,d,e} {a~z}

// Palabras Reservadas
CONJ = "CONJ"
OPERA = "OPERA"
EVALUAR = "EVALUAR"

// Signos (para expresiones)
LLAVE_IZQ = "{"
LLAVE_DER = "}"
PAREN_IZQ = "("
PAREN_DER = ")"
FLECHA = "->"
COMA = ","
PUNTO_Y_COMA = ";"
DOS_PUNTOS = ":"

// operaciones de conjuntos
UNION = "U"
INTERSECCION = "&"
COMPLEMENTO = "^"
DIFERENCIA = "-"


%eofval{
    return new Symbol(ParserSym.EOF);
%eofval}

%%

// Reglas Lexicas
<YYINITIAL> {

    // Comentarios
    {BLANCOS} {}
    {COMENTARIO_LINEA} {}
    {COMENTARIO_MULTILINEA} {}

    // Palabras reservadas
    {CONJ} { Util.Utilidades.agregarToken(yytext(),"CONJ" , yyline, yycolumn); return new Symbol(ParserSym.CONJ,yyline,yycolumn,yytext()); }
    {OPERA} { Util.Utilidades.agregarToken(yytext(),"OPERA" , yyline, yycolumn); return new Symbol(ParserSym.OPERA,yyline,yycolumn,yytext()); }
    {EVALUAR} { Util.Utilidades.agregarToken(yytext(),"EVALUAR" , yyline, yycolumn); return new Symbol(ParserSym.EVALUAR,yyline,yycolumn,yytext()); }

    // Signos
    {PAREN_IZQ} { Util.Utilidades.agregarToken(yytext(),"PAREN_IZQ" , yyline, yycolumn); return new Symbol(ParserSym.PAREN_IZQ,yyline,yycolumn,yytext()); }
    {PAREN_DER} { Util.Utilidades.agregarToken(yytext(),"PAREN_DER" , yyline, yycolumn); return new Symbol(ParserSym.PAREN_DER,yyline,yycolumn,yytext()); }
    {FLECHA} { Util.Utilidades.agregarToken(yytext(),"FLECHA" , yyline, yycolumn); return new Symbol(ParserSym.FLECHA,yyline,yycolumn,yytext()); }
    {COMA} { Util.Utilidades.agregarToken(yytext(),"COMA" , yyline, yycolumn); return new Symbol(ParserSym.COMA,yyline,yycolumn,yytext()); }
    {PUNTO_Y_COMA} { Util.Utilidades.agregarToken(yytext(),"PUNTO_Y_COMA" , yyline, yycolumn); return new Symbol(ParserSym.PUNTO_Y_COMA,yyline,yycolumn,yytext()); }
    {DOS_PUNTOS} { Util.Utilidades.agregarToken(yytext(),"DOS_PUNTOS" , yyline, yycolumn); return new Symbol(ParserSym.DOS_PUNTOS,yyline,yycolumn,yytext()); }
    {LLAVE_IZQ} { Util.Utilidades.agregarToken(yytext(),"LLAVE_IZQ" , yyline, yycolumn); return new Symbol(ParserSym.LLAVE_IZQ,yyline,yycolumn,yytext()); }
    {LLAVE_DER} { Util.Utilidades.agregarToken(yytext(),"LLAVE_DER" , yyline, yycolumn); return new Symbol(ParserSym.LLAVE_DER,yyline,yycolumn,yytext()); }

    // Operaciones de conjuntos
    {UNION} { Util.Utilidades.agregarToken(yytext(),"UNION" , yyline, yycolumn); return new Symbol(ParserSym.UNION,yyline,yycolumn,yytext()); }
    {INTERSECCION} { Util.Utilidades.agregarToken(yytext(),"INTERSECCION" , yyline, yycolumn); return new Symbol(ParserSym.INTERSECCION,yyline,yycolumn,yytext()); }
    {COMPLEMENTO} { Util.Utilidades.agregarToken(yytext(),"COMPLEMENTO" , yyline, yycolumn); return new Symbol(ParserSym.COMPLEMENTO,yyline,yycolumn,yytext()); }
    {DIFERENCIA} { Util.Utilidades.agregarToken(yytext(),"DIFERENCIA" , yyline, yycolumn); return new Symbol(ParserSym.DIFERENCIA,yyline,yycolumn,yytext()); }

    // Componentes lexicos
    {ID} { Util.Utilidades.agregarToken(yytext(),"ID" , yyline, yycolumn); return new Symbol(ParserSym.ID,yyline,yycolumn,yytext()); }
    {INTERV} { Util.Utilidades.agregarToken(yytext(),"INTERV" , yyline, yycolumn); return new Symbol(ParserSym.INTERV,yyline,yycolumn,yytext()); }
    {LISTA} { Util.Utilidades.agregarToken(yytext(),"LISTA" , yyline, yycolumn); return new Symbol(ParserSym.LISTA,yyline,yycolumn,yytext()); }
    {LISTA_CONJ} { Util.Utilidades.agregarToken(yytext(),"LISTA_CONJ" , yyline, yycolumn); return new Symbol(ParserSym.LISTA_CONJ,yyline,yycolumn,yytext()); }
    {LISTA_ELEMENTOS} { Util.Utilidades.agregarToken(yytext(),"LISTA_ELEMENTOS" , yyline, yycolumn); return new Symbol(ParserSym.LISTA_ELEMENTOS,yyline,yycolumn,yytext()); }

    // Manejo de errores
    . { Util.Utilidades.agregarError(yytext(), "ERROR LEXICO", yyline, yycolumn); System.err.println("Error léxico: Caracter inválido <" + yytext() + "> en la línea " + yyline + ", columna " + yycolumn); }
}
```

Analizador Sintáctico: Para este analizador se implementó en la librería Cup que nos ayuda a que el programa encuentre todo el orden que debe de seguir el programa para que el texto sea aceptado. Nos ayuda con los errores sintácticos y se inició con la declaración de los terminales los cuales fueron llamados e importados del Jflex.



```
// Paquete e importaciones
package Analizadores;
import java_cup.runtime.*;
import java.util.ArrayList;
import Util.Conjunto;
import Util.Evaluacion;
import Util.Operacion;
import Util.Utilidades;

// Definición de la clase del parser
class Parser;

// area de codigo -> parser code
parser code
{:

    public ArrayList<ArrayList> info;


    public void syntax_error(Symbol s){
        System.out.println("Error Sintactico en la linea " +
            (s.left) + " y columna " + (s.right) +
            ". No se esperaba el componente: " + (s.value) + ".");

        if (s.value != null){
            Util.Utilidades.agregarError(s.value.toString(), "ERROR SINTACTICO", s.left, s.right);
        }
    }

    public void unrecovered_syntax_error (Symbol s) throws java.lang.Exception{
        System.out.println("Error Sintactico no recuperable en la linea" +
            (s.left) + " y columna " + (s.right) +
            ". No se esperaba el componente: " + (s.value) + ".");

        if (s.value != null){
            Util.Utilidades.agregarError(s.value.toString(), "ERROR SINTACTICO", s.left, s.right);
        }
    }

:}

// Terminales
terminal String ID, INTERV, LISTA, LISTA_CONJ, LISTA_ELEMENTOS, CONJ, OPERA, EVALUAR, LLAVE_IZQ, LLAVE_DER, PAREN_IZQ, PAREN_DER, FLECHA, COMA, PUNTO_Y_COMA, DOS_PUNTOS, UNION, INTERSECCION, COMPLEMENTO, DIFERENCIA;

// No terminales

non terminal programa;
non terminal ArrayList sentencias;
non terminal sentencia;
non terminal conjunto;
non terminal operacion;
non terminal evaluacion;
non terminal lista;
non terminal definicion;
non terminal ArrayList operacionDef;
non terminal operadoresyconjuntos;
non terminal ArrayList operacionComplemento;
non terminal ArrayList operadoresLista;
non terminal operador;
non terminal ArrayList listaComplemento;
non terminal ArrayList conjuntos;
non terminal ArrayList operacionComb;
non terminal ArrayList OperacionFinal;

// Producción inicial
start with programa;

// Producciones

programa ::= LLAVE_IZQ sentencias:a LLAVE_DER {: System.out.println("Programa válido." ); :}
           | error LLAVE_IZQ sentencias:a LLAVE_DER {: System.out.println("Error antes de las llaves. Continuando..."); :}
           | LLAVE_IZQ sentencias:a LLAVE_DER error {: System.out.println("Error después de las llaves. Continuando..."); :}
           | error LLAVE_IZQ sentencias:a LLAVE_DER error {: System.out.println("Errores antes y después de las llaves. Continuando..."); :}
           ;

sentencias ::= sentencias:a sentencia:b     {:RESULT=a; RESULT.add(b);:}
            | sentencia:a                   {:RESULT=new ArrayList(); RESULT.add(a);:}
            ;

sentencia ::= conjunto:c PUNTO_Y_COMA {: System.out.println("Sentencia Conjunto: "+c); RESULT = c; :}
           | operacion:o PUNTO_Y_COMA {: System.out.println("Sentencia Operacion: "+o); RESULT = o; :}
           | evaluacion:e PUNTO_Y_COMA {: System.out.println("Sentencia Evaluacion: "+e); RESULT = e; :}
           | error PUNTO_Y_COMA       {: System.out.println("Error en la sentencia."); :}
           ;

conjunto ::= CONJ DOS_PUNTOS ID:a FLECHA definicion:b  {: RESULT = new Conjunto(a,b.toString()); Util.Utilidades.agregarConjunto(a,b.toString()); :} ; // CONJ a=conjA b=8-4

definicion ::= INTERV:a     {: RESULT = a; :}
            | LISTA:a       {: RESULT = a; :}
            ;

operacion ::= OPERA DOS_PUNTOS ID:a FLECHA OperacionFinal:b  {: RESULT = new Operacion(a,b.toString()); Util.Utilidades.agregarOperacion(a,b.toString()); :} ;

OperacionFinal ::= OperacionFinal:a operadoresyconjuntos:b {:RESULT=a; RESULT.add(b);:}
                 | operadoresyconjuntos:a               {:RESULT=new ArrayList(); RESULT.add(a);:}
                 ;

operadoresyconjuntos ::= operacionDef:a          {:RESULT = a;:}
                       | operacionComplemento:b  {:RESULT = b;:}
                       ;


operacionDef ::= operacionComb:a conjuntos:b {:RESULT=a; RESULT.add(b);:} ;

operacionComb ::= operadoresLista:a listaComplemento:b {:RESULT=a; RESULT.addAll(b);:}
                | listaComplemento:a operadoresLista:b {:RESULT=a; RESULT.addAll(b);:}
                | operadoresLista:a listaComplemento:b operadoresLista:c {:RESULT=a; RESULT.addAll(b); RESULT.addAll(c);:}
                | listaComplemento:a operadoresLista:b listaComplemento:c {:RESULT=a; RESULT.addAll(b); RESULT.addAll(c);:}
                | listaComplemento:a operadoresLista:b listaComplemento:c operadoresLista:d {:RESULT=a; RESULT.addAll(b); RESULT.addAll(c); RESULT.addAll(d);:}
                | operadoresLista:a listaComplemento:b operadoresLista:c listaComplemento:d {:RESULT=a; RESULT.addAll(b); RESULT.addAll(c); RESULT.addAll(d);:}
                | listaComplemento:a operadoresLista:b listaComplemento:c operadoresLista:d listaComplemento:e {:RESULT=a; RESULT.addAll(b); RESULT.addAll(c); RESULT.addAll(d); RESULT.addAll(e);:}
                | operadoresLista:a listaComplemento:b operadoresLista:c listaComplemento:d operadoresLista:e {:RESULT=a; RESULT.addAll(b); RESULT.addAll(c); RESULT.addAll(d); RESULT.addAll(e);:}
                | listaComplemento:a operadoresLista:b listaComplemento:c operadoresLista:d listaComplemento:e operadoresLista:f {:RESULT=a; RESULT.addAll(b); RESULT.addAll(c); RESULT.addAll(d); RESULT.addAll(e); RESULT.addAll(f);:}
                | operadoresLista:a listaComplemento:b operadoresLista:c listaComplemento:d operadoresLista:e listaComplemento:f {:RESULT=a; RESULT.addAll(b); RESULT.addAll(c); RESULT.addAll(d); RESULT.addAll(e); RESULT.addAll(f);:}
                | operadoresLista      :a                  {:RESULT=a;:}
                ;

operadoresLista ::= operadoresLista:a operador:b    {:RESULT=a; RESULT.add(b);:}
                  | operador:a                      {:RESULT=new ArrayList(); RESULT.add(a);:}
                  ;

operacionComplemento ::= listaComplemento:a LISTA_CONJ:b {:RESULT=a; RESULT.add(b);:} ;

listaComplemento ::= listaComplemento:a COMPLEMENTO:b {:RESULT=a; RESULT.add(b);:}
                   | COMPLEMENTO:a                     {:RESULT = new ArrayList(); RESULT.add(a);:}
                   ;

operador ::= UNION:u          {: RESULT = u; :}
           | INTERSECCION:i    {: RESULT = i; :}
           | DIFERENCIA:d      {: RESULT = d; :}
           ;

conjuntos ::= conjuntos:a LISTA_CONJ:b      {:RESULT=a; RESULT.add(b);:}
            | LISTA_CONJ:a                  {:RESULT=new ArrayList(); RESULT.add(a);:}
            ;

evaluacion ::= EVALUAR PAREN_IZQ LISTA_ELEMENTOS:a COMA ID:b PAREN_DER  {: RESULT = new Evaluacion(b,a); Util.Utilidades.agregarEvaluacion(b,a); :}; // EVALUAR b=operacion1 a={1,2,3,4,5}
```

INTERFAZ: Este paquete cuenta con toda la interfaz y todo lo que se utilizó para crearlo como imágenes para los iconos etc.

Util: Este paquete cuenta con todo el código que se uso para realizar operaciones, evaluaciones, graficas y propiedades.

Conjunto: Cuenta con el constructor del conjunto y sus funciones.

Operación: Cuenta con el constructor de la operación y sus funciones.

Evaluación: Cuenta con el constructor de la evaluación y sus funciones.

Errores: Cuenta con el constructor del error y sus funciones.

Tokens: Cuenta con el constructor del token y sus errores.

Utilidades: Se implementaron varias Listas (Errores, Tokens, Conjuntos, Operaciones, Evaluaciones) con sus respectivas funciones necesarias. Se implemento los reportes HTML de errores y tokens. Se manejo una forma de splitear la información para que se guarde correctamente en listas.

Ejecutar: Se implemento la realización de las operaciones por medio una pila y leyendo la entrada al revés. Esto con el objetivo de realizarlo en notación polaca inversa. También se implementó una salida en consola el cual será la evaluación.

Propiedades: Se implemento la realización de las propiedades por medio de una cola y leyendo la entrada igual al revés. Todas las leyes aplicadas las va almacenando en una lista para luego poder mostrarlas en un Json.

PanelGrafica: Se implemento una forma de graficacion de las operaciones. De igual forma se implemento con una pila y leyendo la entrada al revés para que sea en notación polaca inversa. Los conjuntos se almacenan en la pila y cuando encuenta un operador estos 2 se operan. Luego estas graficas se muestra en el JPanel de la interfaz.