UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
DIVISIÓN DE CIENCIAS DE LA INGENIERÍA
CURSO: ORGANIZACIÓN DE LENGUAJES Y COMPILADORES
CATEDRÁTICO: ING. MOISES GRANADOS

PRÁCTICA 1: GENERADOR DE DIAGRAMAS DE FLUJO
MANUAL TECNICO

ESTUDIANTE **CARNÉ:**
Cristian Alejandro Roldán López 202147280

QUETZALTENANGO, 26 DE FEBRERO 2,026

Tecnologías usadas:

Sistema operativo: Ubuntu 24.04 LTS
IDE: Intelliji Idea versión Ultimate
Plugin: Android Studio
Java: Versión 17
minSdk : 24
compileSdk: 36

Archivo flex

```
package analizador.analizador;

import Modelo.TokenError;
import Modelo.ModeloLexer;
import java_cup.runtime.Symbol;
import java.util.ArrayList;

%%

%public
%unicode
%cup
%class Lexer
%line
%column

%{
                private Symbol symbol(int type) {
                    return new Symbol(type, yyline + 1,
yycolumn + 1, yytext());
                }
                private Symbol symbol(int type, Object value) {
                    return new Symbol(type, yyline + 1,
yycolumn + 1, value);
                }

                public static ArrayList<TokenError>
listaErrores = new ArrayList<>();


%}

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

/*EXPRESIONES A UTILIZAR*/

DIGITO = [0-9]
LETTER = [a-zA-Z]
SEPARADORES = [ \n\r\t]+
```

```
ENTERO = {DIGITO}+
DECIMAL = {DIGITO}+\.{DIGITO}+

ID = {LETTER}({LETTER}|{DIGITO}|"_")*

CADENA = \"[^\"]*\"

COMENTARIO = "#"[^(\r\n)]*

HEXCOLOR = "#"([0-9a-fA-F]{6})

FIGURA =
ELIPSE|CIRCULO|PARALELOGRAMO|RECTANGULO|ROMBO|RECTANGULO_REDON
DEADO
FUENTE = ARIAL|TIMES_NEW_ROMAN|COMIC_SANS|VERDANA

%%

/*  IGNORAR  */

{SEPARADORES}    { }
{COMENTARIO}     { }

/*  CONFIGURACIÓN  */

"%%%%"                      { return
symbol(sym.SEPARADORSECCION); }

"%DEFAULT"                  { return symbol(sym.DEFAULT); }

"%COLOR_TEXTO_SI"           { return
symbol(sym.COLOR_TEXTO_SI); }
"%COLOR_SI"                 { return symbol(sym.COLOR_SI); }
"%FIGURA_SI"                { return symbol(sym.FIGURA_SI); }
"%LETRA_SI"                 { return symbol(sym.LETRA_SI); }
"%LETRA_SIZE_SI"            { return symbol(sym.LETRA_SIZE_SI);
}

"%COLOR_TEXTO_MIENTRAS"     { return
symbol(sym.COLOR_TEXTO_MIENTRAS); }
"%COLOR_MIENTRAS"           { return
symbol(sym.COLOR_MIENTRAS); }
```

```
"%FIGURA_MIENTRAS"          { return
symbol(sym.FIGURA_MIENTRAS); }
"%LETRA_MIENTRAS"           { return
symbol(sym.LETRA_MIENTRAS); }
"%LETRA_SIZE_MIENTRAS"      { return
symbol(sym.LETRA_SIZE_MIENTRAS); }

"%COLOR_TEXTO_BLOQUE"       { return
symbol(sym.COLOR_TEXTO_BLOQUE); }
"%COLOR_BLOQUE"             { return symbol(sym.COLOR_BLOQUE);
}
"%FIGURA_BLOQUE"            { return symbol(sym.FIGURA_BLOQUE);
}
"%LETRA_BLOQUE"             { return symbol(sym.LETRA_BLOQUE);
}
"%LETRA_SIZE_BLOQUE"        { return
symbol(sym.LETRA_SIZE_BLOQUE); }

/*  PALABRAS RESERVADAS  */

"INICIO"          { return symbol(sym.INICIO); }
"FIN"             { return symbol(sym.FIN); }
"SI"              { return symbol(sym.SI); }
"ENTONCES"        { return symbol(sym.ENTONCES); }
"MIENTRAS"        { return symbol(sym.MIENTRAS); }
"FINSI"           { return symbol(sym.FINSI); }
"FINMIENTRAS"     { return symbol(sym.FINMIENTRAS); }
"HACER"           { return symbol(sym.HACER); }
"MOSTRAR"         { return symbol(sym.MOSTRAR); }
"LEER"            { return symbol(sym.LEER); }
"VAR"             { return symbol(sym.VAR); }

/* OPERADORES  */

"=="              { return symbol(sym.IGUALIGUAL); }
"!="              { return symbol(sym.DIFERENTE); }
">="              { return symbol(sym.MAYORIGUAL); }
"<="              { return symbol(sym.MENORIGUAL); }

"&&"              { return symbol(sym.AND); }
"||"              { return symbol(sym.OR); }

"="               { return symbol(sym.IGUAL); }
">"               { return symbol(sym.MAYOR); }
```

```
"<"                 { return symbol(sym.MENOR); }

"+"                 { return symbol(sym.SUMA); }
"-"                 { return symbol(sym.RESTA); }
"*"                 { return symbol(sym.MULTI); }
"/"                 { return symbol(sym.DIVISION); }

"!"                 { return symbol(sym.NOT); }

"("                 { return symbol(sym.PARENTESIS_ABRE); }
")"                 { return symbol(sym.PARENTESIS_CIERRA); }

","                 { return symbol(sym.COMA); }
"|"                 { return symbol(sym.PIPE); }


/*  LITERALES  */

{DECIMAL}           { return symbol(sym.DECIMAL, yytext()); }
{ENTERO}            { return symbol(sym.ENTERO, yytext()); }

{CADENA}            { return symbol(sym.CADENA, yytext()); }

{HEXCOLOR}          { return symbol(sym.HEXCOLOR, yytext()); }

{FIGURA}            { return symbol(sym.FIGURA, yytext()); }
{FUENTE}            { return symbol(sym.FUENTE, yytext()); }

/* IDENTIFICADORES  */

{ID}                { return symbol(sym.VARIABLE, yytext()); }

/* ERROR  */
. {
   listaErrores.add(
        new TokenError(
            yytext(),
            yyline + 1,
            yycolumn + 1,
            "Léxico",
            "Símbolo no existe en el lenguaje"
        )
   );
}
```

Archivo Cup

```
package analizador.analizador;

import java_cup.runtime.Symbol;
import java.util.List;
import java.util.ArrayList;
import Modelo.*;
import configuracion.*;
import estilos.*;
import reportes.*;

parser code {:
   public static ArrayList<TokenError> listaErrores = new
ArrayList<>();
   public static List<ReporteOperador> listaOperador = new
ArrayList<>();
   public static List<ReporteEstructurasControl>
listaEstructuras = new ArrayList<>();
   public static int indiceContador = 1;

   @Override
   public void syntax_error(Symbol symbol) {
       listaErrores.add(new TokenError(
           symbol.value != null ? symbol.value.toString() :
"EOF",
           symbol.left + 1,
           symbol.right + 1,
           "Sintáctico",
           "Error de sintaxis"
       ));
   }
:}

/* ==================== TERMINALES ==================== */
terminal INICIO, FIN, SI, ENTONCES, MIENTRAS, FINSI,
FINMIENTRAS;
terminal HACER, MOSTRAR, LEER, VAR;

terminal SUMA, RESTA, MULTI, DIVISION;
terminal IGUAL, IGUALIGUAL, DIFERENTE;
terminal MAYOR, MENOR, MAYORIGUAL, MENORIGUAL;
terminal AND, OR, NOT;

terminal PARENTESIS_ABRE, PARENTESIS_CIERRA;
```

```
terminal ENTERO, DECIMAL;
terminal CADENA, COMENTARIO, ERROR;
terminal VARIABLE;

/* TERMINALES PARA CONFIGURACIÓN */
terminal DEFAULT;
terminal COLOR_TEXTO_SI, COLOR_SI;
terminal FIGURA_SI, LETRA_SI, LETRA_SIZE_SI;

terminal COLOR_TEXTO_MIENTRAS, COLOR_MIENTRAS;
terminal FIGURA_MIENTRAS, LETRA_MIENTRAS, LETRA_SIZE_MIENTRAS;

terminal COLOR_TEXTO_BLOQUE, COLOR_BLOQUE;
terminal FIGURA_BLOQUE, LETRA_BLOQUE, LETRA_SIZE_BLOQUE;

terminal PIPE, COMA, PA, PC;
terminal HEXCOLOR;
terminal FIGURA;
terminal FUENTE;
terminal SEPARADORSECCION;
terminal IGUALCONF;     /* ← se usa solo en configuración */
terminal PORCENTAJE;

/* ====================== NO TERMINALES ======================
*/
non terminal ResultadoAnalisis programa;
non terminal List<Instrucciones> lista;
non terminal Instrucciones sentencia;
non terminal Instrucciones declaracion;
non terminal String valor;
non terminal Instrucciones mostrar;
non terminal Instrucciones leer;
non terminal Instrucciones si;
non terminal Instrucciones mientras;
non terminal Instrucciones asignacion;

non terminal List<ConfiguracionesUniversales>
configuracion_opcional;
non terminal List<ConfiguracionesUniversales>
lista_configuracion;
non terminal ConfiguracionesUniversales config;

non terminal String expresion;
```

```
non terminal String termino;
non terminal String factor;

/* No terminales para configuración */
non terminal Integer indice;
non terminal Float tam_letra;
non terminal Double num_expr;
non terminal Double termino_num;
non terminal Double factor_num;
non terminal String color_val;
non terminal String figura_val;
non terminal String letra_val;


non terminal String condicion;
non terminal String cond_or;
non terminal String cond_and;
non terminal String cond_not;
non terminal String cond_rel;
non terminal String valor_cond;

start with programa;

/*   GRAMATICA */

programa ::= INICIO lista:l FIN configuracion_opcional:c
{:
   if (c != null) {
       for (ConfiguracionesUniversales conf : c) {
           conf.aplicarConfiguracion(l);
       }
   }
   RESULT = new ResultadoAnalisis(l,c);
:}
;

lista ::= lista:l sentencia:s {: l.add(s); RESULT = l; :}
       | sentencia:s
         {:
             List<Instrucciones> nueva = new ArrayList<>();
             nueva.add(s);
             RESULT = nueva;
         :}
;
```

```
sentencia ::= declaracion:d {: RESULT = d; :}
            | mostrar:m     {: RESULT = m; :}
            | leer:l        {: RESULT = l; :}
            | si:s          {: RESULT = s; :}
            | mientras:c        {: RESULT = c; :}
            | asignacion:e       {: RESULT = e; :}
            | error
            {:
                  System.out.println("Error recuperado en
sentencia");
                  RESULT = null;
            :}
            ;

declaracion ::= VAR VARIABLE:v IGUAL valor:val
{: RESULT = new Declaracion(v.toString(), val,
indiceContador++); :};

valor ::= ENTERO:e {: RESULT = e.toString(); :}
       | DECIMAL:d {: RESULT = d.toString(); :};

mostrar ::= MOSTRAR CADENA:c
{: RESULT = new Mostrar(c.toString(), indiceContador++); :};

leer ::= LEER VARIABLE:v
{: RESULT = new Leer(v.toString(), indiceContador++); :};

si ::= SI condicion:c ENTONCES lista:l FINSI
{:
   listaEstructuras.add(new ReporteEstructurasControl(
       "SI",
       1,
       c
   ));
   RESULT = new Si(c, l, indiceContador++);
:}
;

mientras ::= MIENTRAS condicion:c HACER lista:l FINMIENTRAS
{:
   listaEstructuras.add(new ReporteEstructurasControl(
       "MIENTRAS",
        1,
```

```
        c
    ));
    RESULT = new Mientras(c, l, indiceContador++);
:}
;

condicion ::= cond_or:c {: RESULT = c; :}
;


cond_or ::= cond_or:c1 OR cond_and:c2
            {: RESULT = c1 + " || " + c2; :}

          | cond_and:c
            {: RESULT = c; :}
;


cond_and ::= cond_and:c1 AND cond_not:c2
             {: RESULT = c1 + " && " + c2; :}

           | cond_not:c
             {: RESULT = c; :}
;


cond_not ::= NOT cond_not:c
             {: RESULT = "!" + c; :}

           | PARENTESIS_ABRE cond_or:c PARENTESIS_CIERRA
             {: RESULT = "(" + c + ")"; :}

           | cond_rel:c
             {: RESULT = c; :}
;


cond_rel ::= valor_cond:v1 MENOR valor_cond:v2
             {:
                 RESULT = v1 + " < " + v2;
             :}

           | valor_cond:v1 MAYOR valor_cond:v2
             {: RESULT = v1 + " > " + v2; :}
```

```
          | valor_cond:v1 IGUALIGUAL valor_cond:v2
            {: RESULT = v1 + " == " + v2; :}

          | valor_cond:v1 DIFERENTE valor_cond:v2
            {: RESULT = v1 + " != " + v2; :}

          | valor_cond:v1 MAYORIGUAL valor_cond:v2
            {: RESULT = v1 + " >= " + v2; :}

          | valor_cond:v1 MENORIGUAL valor_cond:v2
            {: RESULT = v1 + " <= " + v2; :}
;


valor_cond ::= VARIABLE:v {: RESULT = v.toString(); :}
             | ENTERO:e   {: RESULT = e.toString(); :}
             | DECIMAL:d  {: RESULT = d.toString(); :}
;

asignacion ::= VARIABLE:v IGUAL expresion:o
{: RESULT = new Asignacion(v.toString(), o, indiceContador++);
:};

expresion ::= expresion:e1 SUMA termino:t
            {:
                listaOperador.add(new ReporteOperador(
                    "Suma",
                    e1left + 1,
                    tleft,
                    e1 + " + " + t
                ));
                RESULT = e1 + " + " + t;
            :}
          | expresion:e1 RESTA termino:t
            {:
                listaOperador.add(new ReporteOperador(
                    "Resta",
                    e1left + 1,
                    tleft,
                    e1 + " - " + t
                ));
                RESULT = e1 + " - " + t;
            :}
```

```
          | termino:t {: RESULT = t; :}
          ;

termino ::= termino:t1 MULTI factor:f
            {:
                listaOperador.add(new ReporteOperador(
                    "Multiplicación",
                    t1left + 1,
                    fleft,
                    t1 + " * " + f
                ));
                RESULT = t1 + " * " + f;
            :}
          | termino:t1 DIVISION factor:f
            {:
                listaOperador.add(new ReporteOperador(
                    "División",
                    t1left + 1,
                    fleft,
                    t1 + " / " + f
                ));
                RESULT = t1 + " / " + f;
            :}
          | factor:f {: RESULT = f; :}
          ;

factor ::= PARENTESIS_ABRE expresion:e PARENTESIS_CIERRA {:
RESULT = "(" + e + ")"; :}
         | ENTERO:e    {: RESULT = e.toString(); :}
         | DECIMAL:d   {: RESULT = d.toString(); :}
         | VARIABLE:v {: RESULT = v.toString(); :};

configuracion_opcional ::= SEPARADORSECCION
lista_configuracion:l {: RESULT = l; :}
                        | {: RESULT = null; :};

lista_configuracion ::= lista_configuracion:l config:c
                       {: l.add(c); RESULT = l; :}

                     | lista_configuracion:l error
                       {:
                           System.out.println("Error en
configuración, recuperando...");
                           RESULT = l;
```

```
                        :}

                    | config:c
                      {:
                          List<ConfiguracionesUniversales>
nueva = new ArrayList<>();
                          nueva.add(c);
                          RESULT = nueva;
                        :};

indice ::= ENTERO:i {: RESULT =
Integer.parseInt(i.toString()); :};

tam_letra ::= num_expr:n {: RESULT = n.floatValue(); :};

figura_val ::= FIGURA:f {: RESULT =
f.toString().toUpperCase(); :};
letra_val  ::= FUENTE:f {: RESULT =
f.toString().toUpperCase(); :};

num_expr ::= num_expr:e1 SUMA termino_num:t {: RESULT = e1 +
t; :}
           | num_expr:e1 RESTA termino_num:t {: RESULT = e1 -
t; :}
           | termino_num:t {: RESULT = t; :};

termino_num ::= termino_num:t1 MULTI factor_num:f {: RESULT =
t1 * f; :}
              | termino_num:t1 DIVISION factor_num:f {: RESULT
= t1 / f; :}
              | factor_num:f {: RESULT = f; :};

factor_num ::= PARENTESIS_ABRE num_expr:e PARENTESIS_CIERRA {:
RESULT = e; :}
             | ENTERO:e    {: RESULT =
Double.parseDouble(e.toString()); :}
             | DECIMAL:d   {: RESULT =
Double.parseDouble(d.toString()); :};

color_val ::= HEXCOLOR:h {: RESULT = h.toString(); :}
            | num_expr:r COMA num_expr:g COMA num_expr:b {:
                int rr = (int) Math.round(r);
                int gg = (int) Math.round(g);
                int bb = (int) Math.round(b);
```

```
                RESULT = rr + "," + gg + "," + bb;
              :};

config ::= DEFAULT IGUAL indice:i
          {: RESULT = new Default(i); :}

        | COLOR_TEXTO_SI IGUAL color_val:c PIPE indice:i
        {: RESULT = new
ColorTextoSi(ConfiguracionColor.Companion.parse(c), i); :}

        | COLOR_SI IGUAL color_val:c PIPE indice:i
        {: RESULT = new
ConfiguracionColorSi(ConfiguracionColor.Companion.parse(c),
i); :}

        | FIGURA_SI IGUAL figura_val:f PIPE indice:i
        {:RESULT = new
ConfiguracionFiguraSi(TipoFigura.valueOf(f), i);:}

        | LETRA_SI IGUAL letra_val:l PIPE indice:i
        {: RESULT = new
ConfiguracionLetraSi(LetraFuente.valueOf(l), i); :}

        | LETRA_SIZE_SI IGUAL tam_letra:s PIPE indice:i
        {: RESULT = new ConfiguracionTamaLetraSi(s, i); :}

        /* MIENTRAS */
        |  COLOR_TEXTO_MIENTRAS IGUAL color_val:c PIPE
indice:i
        {: RESULT = new
ColorTextoMientras(ConfiguracionColor.Companion.parse(c), i);
:}
        |  COLOR_MIENTRAS IGUAL color_val:c PIPE indice:i
        {: RESULT = new
ConfiguracionColorMientras(ConfiguracionColor.Companion.parse(
c), i); :}
        |  FIGURA_MIENTRAS IGUAL figura_val:f PIPE indice:i
        {: RESULT = new
ConfiguracionFiguraMientras(TipoFigura.valueOf(f), i); :}
        | LETRA_MIENTRAS IGUAL letra_val:l PIPE indice:i
        {: RESULT = new
ConfiguracionLetraMientras(LetraFuente.valueOf(l),i); :}
        | LETRA_SIZE_MIENTRAS IGUAL tam_letra:s PIPE indice:i
```

```
        {: RESULT = new ConfiguracionTamaLetraMientras(s, i);
:}

        /* BLOQUES DE CODIGO */
        |  COLOR_TEXTO_BLOQUE IGUAL color_val:c PIPE indice:i
        {: RESULT = new
ColorTextoBloque(ConfiguracionColor.Companion.parse(c), i); :}
        |  COLOR_BLOQUE IGUAL color_val:c PIPE indice:i
        {: RESULT = new
ConfiguracionColorBloque(ConfiguracionColor.Companion.parse(c)
, i); :}
        |  FIGURA_BLOQUE IGUAL figura_val:f PIPE indice:i
        {: RESULT = new
ConfiguracionFiguraBloque(TipoFigura.valueOf(f), i); :}
        | LETRA_BLOQUE IGUAL letra_val:l PIPE indice:i
        {: RESULT = new
ConfiguracionLetraBloque(LetraFuente.valueOf(l),i); :}
        | LETRA_SIZE_BLOQUE IGUAL tam_letra:s PIPE indice:i
        {: RESULT = new ConfiguracionTamaLetraBloque(s, i); :}
        ;
```

Clases utilizadas para reconocer las palabras reservadas

```
package Modelo

import configuracion.LetraFuente
import configuracion.TipoFigura
import java.io.Serializable


abstract class Instrucciones(open val indice : Int, open var
colorTexto: Int? = null,
                             open var colorFondo: Int? = null, open
var figura: TipoFigura = TipoFigura.RECTANGULO,
                             open var tipoLetra: LetraFuente =
LetraFuente.ARIAL, open var tamLetra: Float = 36f) : Serializable {


   open fun obtenerSubInstrucciones(): List<Instrucciones>? = null
}
```

```
package Modelo

class Asignacion(val varibale : String, val expresion : String,
override val indice: Int) : Instrucciones(indice) {

   override fun toString() = "$varibale = $expresion"
}
```

```
package Modelo

class Condicion(val variable : String, val signo : String, val
variableCierre: String)  {

   override fun toString() = "($variable $signo $variableCierre)"
}
```

```
package Modelo

import configuracion.LetraFuente
import configuracion.TipoFigura


class Declaracion(val variable : String, val valor : String,
```

```
                override val indice: Int) : Instrucciones(indice) {

    override fun toString() = "$variable = $valor"
}
```

```
package Modelo

class Leer(val variable : String, override val indice: Int) :
Instrucciones(indice) {

    override fun toString() = "$variable"
}
```

```
package Modelo

import configuracion.LetraFuente
import configuracion.TipoFigura


class Mientras(val condicion: String, val cuerpo:
List<Instrucciones>, override val indice: Int)
    : Instrucciones(indice, figura = TipoFigura.ROMBO) {

    override fun toString() = "$condicion"

    fun getTextoCondicion(): String {
        return "¿$condicion?"
    }

    override fun obtenerSubInstrucciones(): List<Instrucciones>? {
        return cuerpo
    }
}
```

```
package Modelo




class Mostrar(val texto : String, override val indice: Int) :
Instrucciones(indice) {

    override fun toString() = "$texto"
}
```

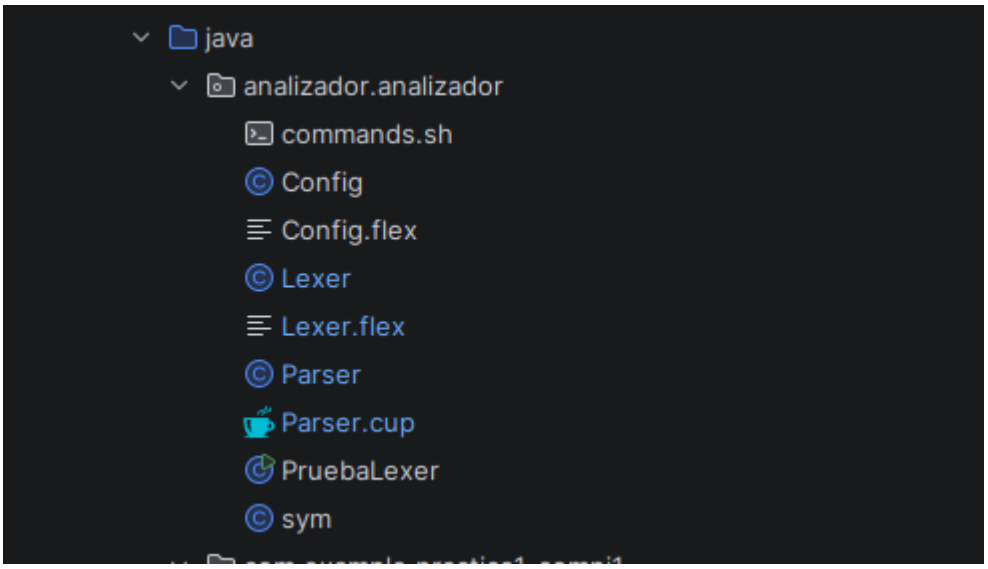
```
package Modelo
```

```
import configuracion.LetraFuente
import configuracion.TipoFigura


class Si(val condicion : String, val cuerpoCondicion :
List<Instrucciones>, override val indice: Int)
    : Instrucciones(indice, figura = TipoFigura.ROMBO) {

    override fun toString() = "$condicion"

    fun getTextoCondicion(): String {
        return "¿$condicion?"
    }

    override fun obtenerSubInstrucciones(): List<Instrucciones>? {
        return cuerpoCondicion
    }
}
```

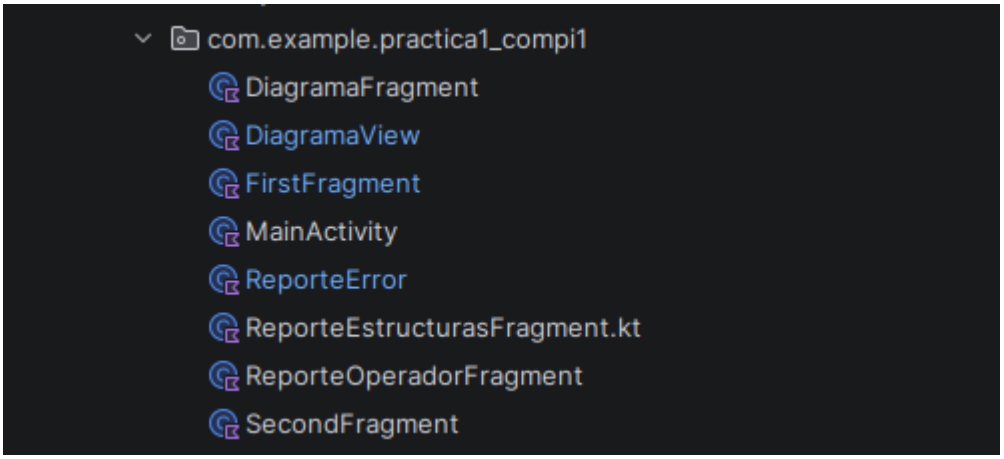
Las clases heredan de Instrucción que es una clase abstracta y cada clase hija recibe distintos parámetros para poder funcionar.

Organización del proyecto

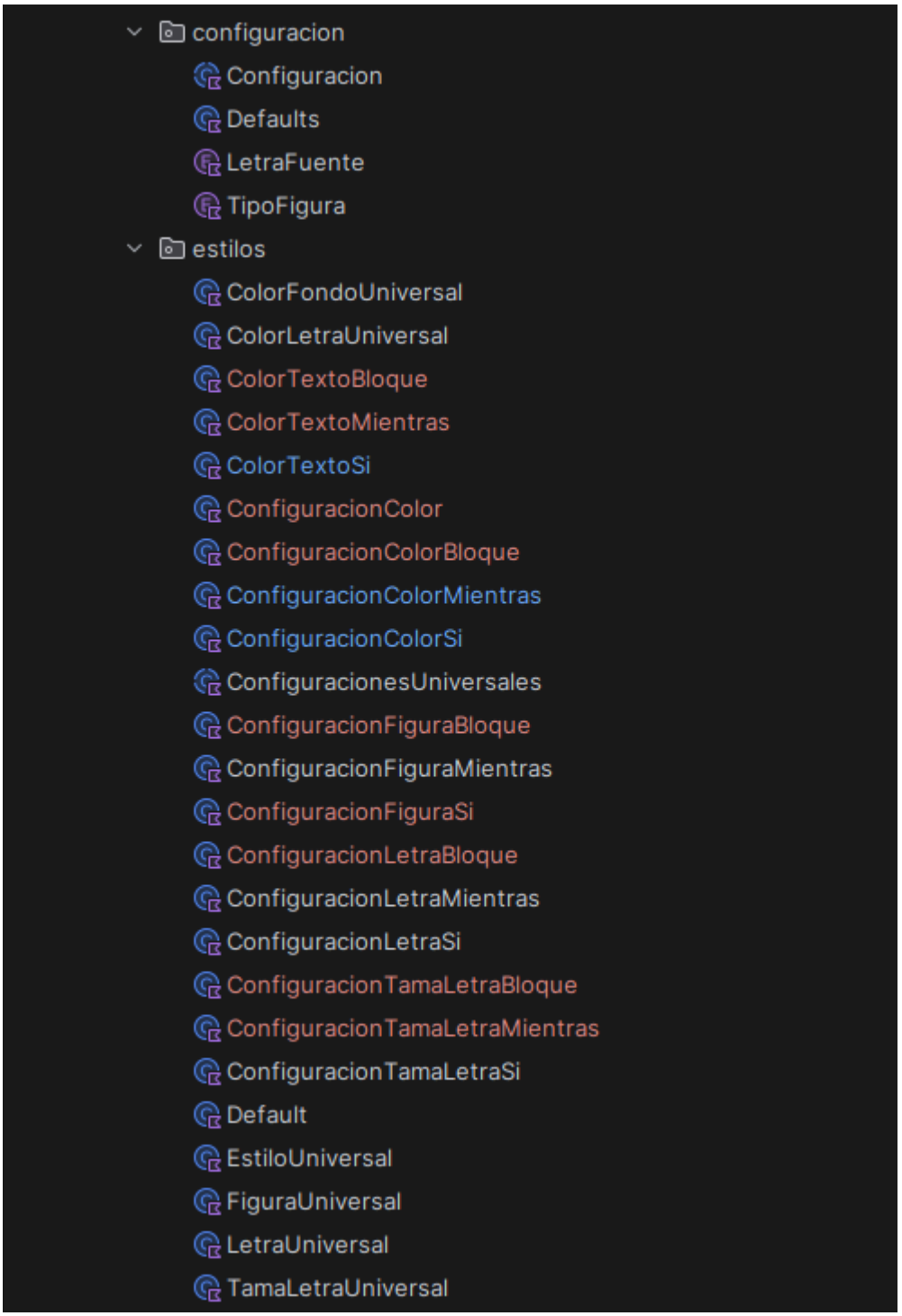
Clases encargadas de generar el analizador Lexico y Sintáctico



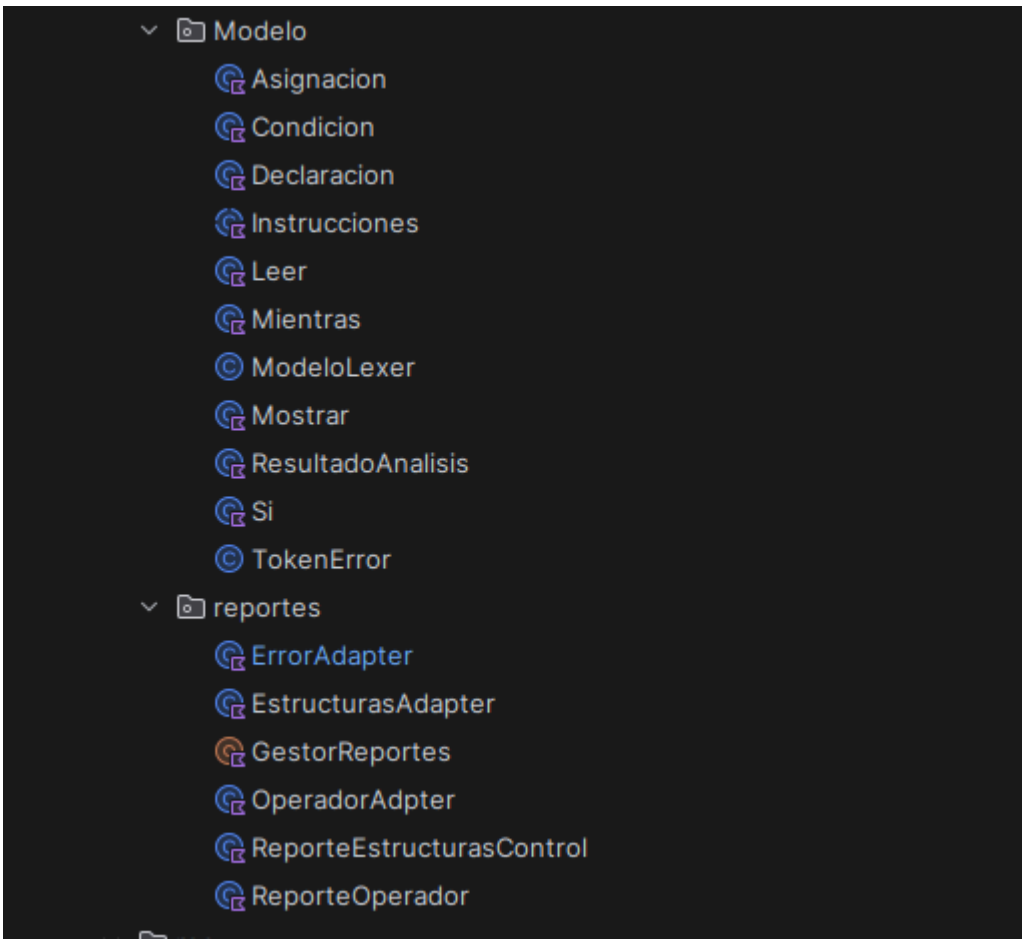
Clases encargadas de recibir y realizar el análisis de las instrucciones que se mandan.



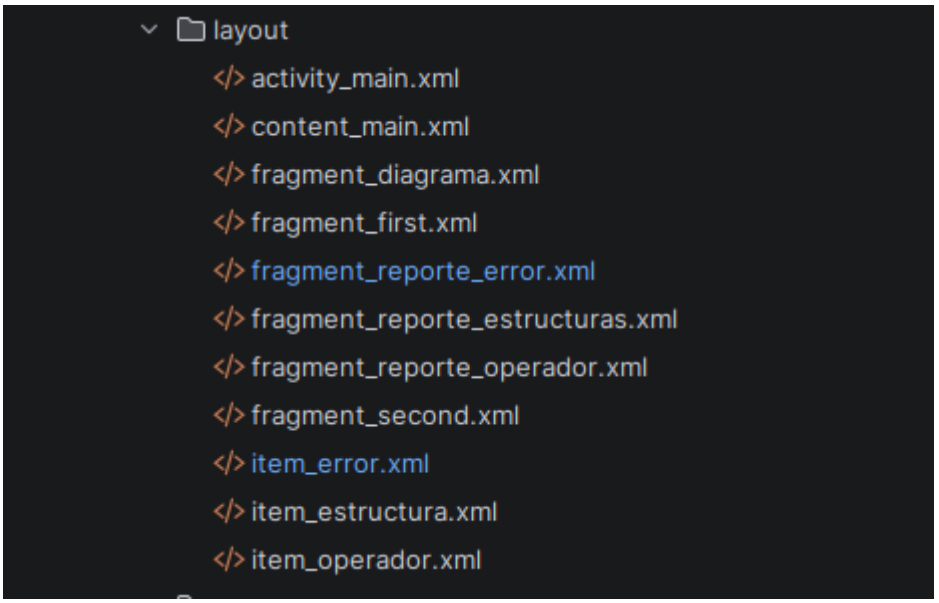
Clases encargadas de la configuración que dara el usuario si así lo desea.



Clases que se encargan de los modelos de las palabras reservadas y la realización de los atributos para los reportes y errores



Archivos .xml que son donde se realiza la interfaz del usuario



Archivo que se configura para navegar entre pestañas