# Manual Técnico



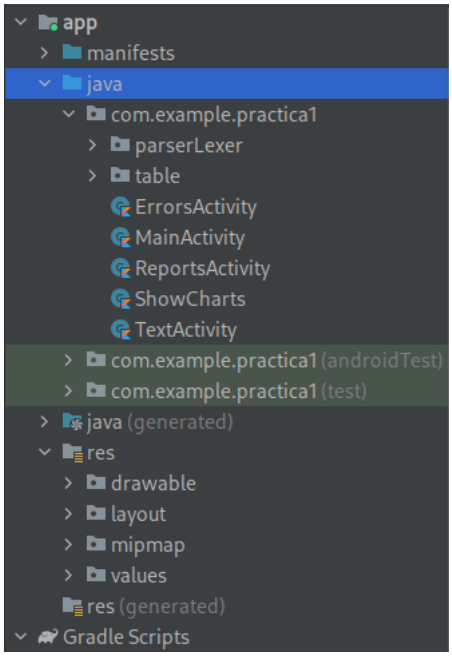
## Información del sistema:

- OS: `Arch Linux`
- Kernel: `x86_64 Linux 5.16.8-arch1-1`
- CPU: `Intel Core i3-4005U @ 4x 1.7GHz`
- GPU: `Intel Corporation Haswell-ULT Integrated Graphics Controller`
- RAM: `7881MiB`
- Versión de java: `11.0.13`
- IDE: `android-studio 2021.1.1.22-1`
- Control de versiones: `git version 2.35.1`

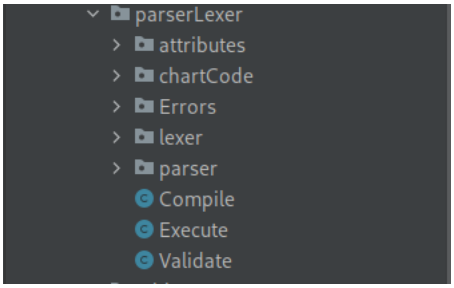
## Especificaciones del proyecto

```
android {
    compileSdk 32

    defaultConfig {
        applicationId "com.example.practica1"
        minSdk 22
        targetSdk 32
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
```

## Estuctura del proyecto




El proyecto consta de dos partes muy importantes. La primera es la que se encarga del lenguaje, reconocerlo y compilarlo. Usando las herramientas de `jflex` y `java cup` se obtienen las clases encargadas del lenguaje, pero además en esta parte se usan clases guardadas en el paquete `parserLexer`:

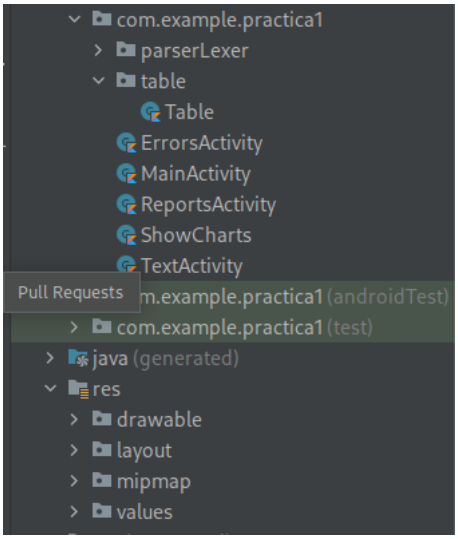



Estas clases ayudan a ejecutar cada instrucción, desde su validación hasta el reconocimiento de errores `semánticos`. En el paquete `attributes` se guardan las clases que guardan los los atributos de cada gráfica (obtenidos directamente desde la gramática).

La gramática, aunque detecta errores no es capaz de reconocer todos. Es por eso que se usa la clase `Validate.java` para validar los atributos de las gráficas depués de ser obtenidas. Desde validar que vengar los atributos correctos hasta que el resultado de las operaciones sea el matemáticamente correcto.

Si se detecta un error, se guardan en la clase `Error` que está dentro del paquete `Errors`. En esta clase están dos clases más que son encargadas de crear objetos que guarden las información del error. El error es almacenado y posteriormente mostrado.

En el paquete `chartCode` están las clases que guardan los datos, previamente validados, de las gráficas y los convierten a una forma que pueda entender la libreria `MPAndroidChart`




Para manejar las vistas en la apliación de android se usan las actividades mostradas en la imagen anterior. Los archivos de XML, como en toda aplicación android, se guardan el la carpeta `res`, esta carpeta contiene la configuración y layout de todas las actividades. Todas las actividades están escritas en `Kotlin`

# Herramientas y librerías

## JFlex

JFlex es un generador de analizadores léxicos, (también conocido como un generador de scanners) para Java y escrito en Java

- versión: `1.8.2`

JFlex

JFlex is a lexical analyzer generator (also known as scanner generator) for Java, written in Java. A lexical analyzer generator takes as input a specification with a set of regular expressions and corresponding actions. It generates a program (a lexer) that reads input, matches the input against the regular expressions in the spec

https://www.jflex.de/


## CUP

Construcción de analizadores útiles, por sus siglas en inglés (Construction of Useful Parser). Es un generador de analizadores de tipo `LALR` escrito en Java. Desarrollado por C. Scott Ananian, Frank Flannery, Dan Wang, Andrew W. Appel y Michael Petter.

- versión: `v0.11b 20160615`

CUP

CUP stands for Construction of Useful Parsers and is an LALR parser generator for Java. It was developed by C. Scott Ananian, Frank Flannery, Dan Wang, Andrew W. Appel and Michael Petter. It implements standard LALR(1) parser generation.

http://www2.cs.tum.edu/projects/cup/


## MPAndroidChart

Es una librería para la creación de gráficos en Android. Gráficos de barras, de pastel, de líneas entre otros.

- versión: `v3.0.3`

GitHub - PhilJay/MPAndroidChart: A powerful 🚀 Android chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts a

A powerful 🚀 Android chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts as well as scaling, panning and animations. - GitHub - Ph
chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts as well as scaling, panning and animations.

https://github.com/PhilJay/MPAndroidChart

# Definición de tokens (expresiones regulares)

## Palabras reservadas

- **Def** o **def:** `([Dd]ef)`
- **Barras:** `(Barras)`
- **Pie:** `(Pie)`
- **titulo:** `(titulo)`
- **ejex:** `(ejex)`
- **ejey:** `(ejey)`
- **etiquetas:** `(etiquetas)`
- **valores:** `(valores)`
- **unir:** `(unir)`
- **tipo:** `(tipo)`
- **total:** `(total)`
- **extra:** `(extra)`
- **Ejecutar:** `(Ejecutar)`
- **Cantidad:** `(Cantidad)`
- **Porcentaje:** `(Porcentaje)`

## Operadores aritméticos y número

- Suma: `[+]`
- Resta: `[-]`
- Multiplicación: `[*]`
- División: `[\/]`
- Paréntesis abrir: `[\(]`
- Paréntesis cerrar: `[\)]`
- Números enteros: `0|[1-9][0-9]*`
- Números: `0|[1-9][0-9]*\.\d+`

## De uso en estructuras

- Corchetes abrir: `[\[]`
- Corchetes cerrar: `[\]]`
- LLaves abrir: `[\{]`
- Llaves cerrar: `[\}]`
- Dos puntos: `[:]`

- Coma: `[,]`
- Punto y coma: `[;]`

## Comentarios:

- Comentario de una línea: `(#[^\r\n]*)`

## Cadenas de caracteres

- Strings: `(\"[^ \"\n\t\f\r]*\")`

## JFLEX

```
%class LexerP1
%public
%unicode
%line
%column
%cup
//%standalone


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

LineTerminator = \r|\n|\r\n
WhiteSpace = {LineTerminator} | [ \t\f]

/* Coments */
InputCharacter = [^\r\n]
Comment = "#" {InputCharacter}* {LineTerminator}?

/* Reserved words */

Def = ([Dd]ef)
Barras = (Barras)
Pie = (Pie)
titulo = (titulo)
ejex = (ejex)
ejey = (ejey)
etiquetas = (etiquetas)
valores = (valores)
unir = (unir)
tipo = (tipo)
total = (total)
extra = (extra)
Ejecutar = (Ejecutar)
Cantidad = (Cantidad)
Porcentaje = (Porcentaje)

/* Maths */

Plus = [+]
Minus = [-]
Times = [*]
Division = [\/]
LParen = [\(]
RParen = [\)]
Integer = 0|[1-9][0-9]*
Decimal = {Integer} \. \d+

/* Structures */

LBracket = [\[]
RBracket = [\]]
LBrace = [\{]
RBrace = [\}]
Colon = [:]
Comma = [,]
Semicolon = [;]

/* Strings */
```

```
Q = [\"]
StringContent = ([^\"\n\r]|[\n])*
String = {Q}{StringContent}{Q}

%%

{Comment}
{/* Ignorar */}
{String}
{return new Symbol(sym.STRING , yyline + 1, yycolumn + 1, yytext());}
{Def}
{return new Symbol(sym.DEF , yyline + 1, yycolumn + 1, yytext());}
{Barras}
{return new Symbol(sym.BARRAS , yyline + 1, yycolumn + 1, yytext());}
{Pie}
{return new Symbol(sym.PIE , yyline + 1, yycolumn + 1, yytext());}
{titulo}
{return new Symbol(sym.TITULO , yyline + 1, yycolumn + 1, yytext());}
{ejex}
{return new Symbol(sym.EJEX , yyline + 1, yycolumn + 1, yytext());}
{ejey}
{return new Symbol(sym.EJEY , yyline + 1, yycolumn + 1, yytext());}
{etiquetas}
{return new Symbol(sym.ETIQUETAS , yyline + 1, yycolumn + 1, yytext());}
{valores}
{return new Symbol(sym.VALORES , yyline + 1, yycolumn + 1, yytext());}
{unir}
{return new Symbol(sym.UNIR , yyline + 1, yycolumn + 1, yytext());}
{tipo}
{return new Symbol(sym.TIPO , yyline + 1, yycolumn + 1, yytext());}
{total}
{return new Symbol(sym.TOTAL , yyline + 1, yycolumn + 1, yytext());}
{extra}
{return new Symbol(sym.EXTRA , yyline + 1, yycolumn + 1, yytext());}
{Ejecutar}
{return new Symbol(sym.EJECUTAR , yyline + 1, yycolumn + 1, yytext());}
{Cantidad}
{return new Symbol(sym.CANTIDAD , yyline + 1, yycolumn + 1, yytext());}
{Porcentaje}
{return new Symbol(sym.PORCENTAJE , yyline + 1, yycolumn + 1, yytext());}

{Plus}
{return new Symbol(sym.PLUS , yyline + 1, yycolumn + 1, yytext());}
{Minus}
{return new Symbol(sym.MINUS , yyline + 1, yycolumn + 1, yytext());}
{Times}
{return new Symbol(sym.TIMES , yyline + 1, yycolumn + 1, yytext());}
{Division}
{return new Symbol(sym.DIVISION , yyline + 1, yycolumn + 1, yytext());}
{LParen}
{return new Symbol(sym.LPAREN , yyline + 1, yycolumn + 1, yytext());}
{RParen}
{return new Symbol(sym.RPAREN , yyline + 1, yycolumn + 1, yytext());}
{Integer}
{return new Symbol(sym.INTEGER , yyline + 1, yycolumn + 1, new Double(yytext()));}
{Decimal}
{return new Symbol(sym.DECIMAL , yyline + 1, yycolumn + 1, new Double(yytext()));}

{LBracket}
{return new Symbol(sym.LBRACKET , yyline + 1, yycolumn + 1, yytext());}
{RBracket}
{return new Symbol(sym.RBRACKET , yyline + 1, yycolumn + 1, yytext());}
{LBrace}
{return new Symbol(sym.LBRACE , yyline + 1, yycolumn + 1, yytext());}
{RBrace}
{return new Symbol(sym.RBRACE , yyline + 1, yycolumn + 1, yytext());}
{Colon}
{return new Symbol(sym.COLON , yyline + 1, yycolumn + 1, yytext());}
{Comma}
{return new Symbol(sym.COMMA , yyline + 1, yycolumn + 1, yytext());}
{Semicolon}
{return new Symbol(sym.SEMICOLON , yyline + 1, yycolumn + 1, yytext());}

{WhiteSpace} {/* Ignorar */}

[\^´°¬|_!$%&=?'¡¿\w]+ {
                        Errors.getErrors().addLS(yyline+1, yycolumn+1, "Cadena no definida", yytext(), Errors.LEXICAL);
                    }

[^] {
```

```
        Errors.getErrors().addLS(yyline+1, yycolumn+1, "Caracter inesperado", yytext(), Errors.LEXICAL);
    }
```

# Gramática

## Reconocimineto básico

```
s::=
      chart s
      |execute s
      |chart
      |execute
      ;
```

## Gráficos

```
chart::=
          DEF BARRAS LBRACE cuerpo_barras RBRACE
          |DEF PIE LBRACE cuerpo_pie RBRACE
          ;
```

```
cuerpo_barras::=
                        statement_barras:att fin cuerpo_barras:arr
                        |statement_barras:att fin
                        ;

statement_barras::=
                        titulo:ti
                        |ejex:ex
                        |ejey :ey
                        |unir:u
                        ;
```

```
cuerpo_pie::=
                        statement_pie fin cuerpo_pie:arr
                        | statement_pie fin
                        ;
statement_pie::=
                        titulo
                        |etiquetas
                        |valores
                        |unir
                        |tipo
                        |total
                        |extra
                        ;
```

## Jaca CUP

```
parser code {:
    public ParserP1(LexerP1 lexer){
        super(lexer);
    }


    public void syntax_error(Symbol cur_token) {
        HashMap<String, String> symbolNames = Validate.getSymbolNames();
    List<Integer> tokens = expected_token_ids();
        int line = cur_token.left;
        int column = cur_token.right;
        String lexeme = cur_token.value.toString();
    int type = Errors.SINTAX;
```

```
        String des = "Se esperaba: ";
    for(Integer i : tokens) {
            String fromId = symbl_name_from_id(i);
            String symbolName = symbolNames.get(fromId);
            if(symbolName != null){
                des += "'"+symbolName+ "' o ";
            }
            else{
                des += fromId +" o ";
            }
    }
        des = des.substring(0, des.length() - 3);
        Errors.getErrors().addLS(line, column, des, lexeme, type);
  }

    public void report_fatal_error(String message, Object info) {
    System.out.println("message: " + message);
    System.out.println("info: " + info);
  }

    protected int error_sync_size() {
        return 1;
    }
:}


terminal SEMICOLON, COLON, COMMA;
terminal LBRACE, RBRACE;
terminal LBRACKET, RBRACKET;
terminal LPAREN, RPAREN;
terminal String STRING;

terminal PLUS, MINUS, DIVISION, TIMES;
terminal Double INTEGER, DECIMAL;
// Graphs definition
terminal DEF, BARRAS, PIE;
terminal TITULO, EJEX, EJEY, UNIR;
terminal ETIQUETAS, VALORES, TIPO, EXTRA, TOTAL;
terminal String CANTIDAD, PORCENTAJE;
terminal ERROR;

terminal EJECUTAR;


non terminal s, fin;
non terminal ChartCode chart;

non terminal ArrayList<Attribute> cuerpo_barras;
non terminal Attribute statement_barras;
non terminal ArrayList<Attribute> cuerpo_pie;
non terminal Attribute statement_pie;

non terminal Attribute<String> titulo;
non terminal Attribute<ArrayNode<String>> ejex;
non terminal Attribute<ArrayNode<Operation>> ejey;
non terminal Attribute<ArrayNode<Operation[]>> unir;
non terminal Attribute<ArrayNode<String>> etiquetas;
non terminal Attribute<ArrayNode<Operation>> valores;
non terminal Attribute<String> tipo;
non terminal Attribute<Operation> total;
non terminal Attribute<String> extra;
non terminal String execute;

non terminal String pie_types;

non terminal ArrayNode<String>array_strings;
non terminal ArrayNode<String>strings_comma;
non terminal ArrayNode<Operation> array_operations;
non terminal ArrayNode<Operation> operations_comma;
non terminal ArrayNode<Operation[]> tuplas_comma;
non terminal Operation[] tupla;

non terminal Operation operation;
non terminal Operation factor;
non terminal Operation term;
```

```
s::=
                        chart:cht s
                        {:
                            if(cht != null) Execute.getExecute().addChart(cht);
                        :}
                        |execute:ex s
                        {:
                            if(ex != null) Execute.getExecute().addExecuteCall(ex);
                        :}
                        | chart:cht
                        {:
                            if(cht != null) Execute.getExecute().addChart(cht);
                        :}
                        | execute:ex
                        {:
                            if(ex != null) Execute.getExecute().addExecuteCall(ex);
                        :}
                        ;

fin::=
                        SEMICOLON
                        ;

chart::=
                        DEF BARRAS LBRACE cuerpo_barras:arr RBRACE
                        {:
                            BarChartCode chart = Validate.validateBar(arr);
                            RESULT = chart;
                        :}
                        |DEF PIE LBRACE cuerpo_pie:arr RBRACE
                        {:
                            PieChartCode chart = Validate.validatePie(arr);
                            RESULT = chart;
                        :}
                        ;

cuerpo_barras::=
                        statement_barras:att fin cuerpo_barras:arr
                        {:
                            arr.add(att);
                            RESULT = arr;
                        :}
                        |statement_barras:att fin
                        {:
                            ArrayList<Attribute> attributes = new ArrayList<>();
                            attributes.add(att);
                            RESULT = attributes;
                        :}
                        ;
statement_barras::=
                        titulo:ti
                        {:
                            RESULT = ti;
                        :}
                        |ejex:ex
                        {:
                            RESULT = ex;
                        :}
                        |ejey :ey
                        {:
                            RESULT = ey;
                        :}
                        |unir:u
                        {:
                            RESULT = u;
                        :}
                        ;

titulo::=
                        TITULO COLON STRING:st
                        {:
                          st = st.replaceAll("\"", "");
                          Attribute<String> att = new Attribute<>(st, Attribute.TITULO, stleft);
                          RESULT = att;
                        :}
                        ;
ejex::=
                        EJEX COLON array_strings:arr
                        {:
                            Attribute<ArrayNode<String>> att = new Attribute<>(arr, Attribute.EJEX, arrleft);
```

```
                        RESULT = att;
                    :}
                    ;

array_strings::=
                    LBRACKET strings_comma:arr RBRACKET
                    {:
                        RESULT = arr;
                    :}
                    ;
strings_comma::=
                    STRING:st COMMA strings_comma:nexts
                    {:
                        st = st.replaceAll("\"", "");
                        ArrayNode<String> node = new ArrayNode<String>(st);
                        node.setNext(nexts);
                        RESULT = node;
                    :}
                    |STRING:st
                    {:
                        st = st.replaceAll("\"", "");
                        ArrayNode<String> node = new ArrayNode<String>(st);
                        RESULT = node;
                    :}
                    ;
ejey::=
                    EJEY COLON array_operations:arr
                    {:
                        Attribute<ArrayNode<Operation>> att = new Attribute<>(arr, Attribute.EJEY, arrleft);
                        RESULT = att;
                    :}
                    ;

array_operations::=
                    LBRACKET operations_comma:arr RBRACKET
                    {:
                        RESULT = arr;
                    :}
                    ;
operations_comma::=
                    operation:o COMMA operations_comma:nexts
                    {:
                        ArrayNode<Operation> node = new ArrayNode<Operation>(o);
                        node.setNext(nexts);
                        RESULT = node;
                    :}
                    | operation: o
                    {:
                        ArrayNode<Operation> node = new ArrayNode<Operation>(o);
                        RESULT = node;
                    :}
                    ;
unir::=
                    UNIR COLON LBRACKET tuplas_comma:tuplas RBRACKET
                    {:
                       Attribute<ArrayNode<Operation[]>> att = new Attribute<>(tuplas, Attribute.UNIR, tuplasleft);
                       RESULT = att;
                    :}
                    ;
tuplas_comma::=
                    tupla:tp COMMA tuplas_comma:prevs
                    {:
                        ArrayNode<Operation[]> node = new ArrayNode<Operation[]>(tp);
                        node.setNext(prevs);
                        RESULT = node;
                    :}
                    | tupla:tp
                    {:
                        ArrayNode<Operation[]> node = new ArrayNode<Operation[]>(tp);
                        RESULT = node;
                    :}
                    ;
tupla::=
                    LBRACE operation:x COMMA operation:y RBRACE
                    {:
                        Operation[] array = new Operation[2];
                        array[0] = x;
                        array[1] = y;
                        RESULT = array;
                    :}
                    ;
```

```
operation::=
                        operation:o PLUS:sm term:t
                        {:
                            Execute.getExecute().addMathSymbol(smleft, smright, MathSymbol.SUMA);
                            RESULT=new Operation("+", o, t);
                        :}
                        |operation:o MINUS:sm term:t
                        {:
                            Execute.getExecute().addMathSymbol(smleft, smright, MathSymbol.RESTA);
                            RESULT=new Operation("-", o, t);
                        :}
                        |term:t
                        {:RESULT=t;:}
                        ;

term::=
                        term:t TIMES:sm factor:f
                        {:
                            Execute.getExecute().addMathSymbol(smleft, smright, MathSymbol.MULTIPLICACION);
                            RESULT=new Operation("*", t, f);
                        :}
                        |term:t DIVISION:sm factor:f
                        {:
                            Execute.getExecute().addMathSymbol(smleft, smright, MathSymbol.DIVISION);
                            RESULT=new Operation("/", t, f);
                        :}
                        |factor:f
                        {: RESULT=f;:}
                        ;
factor::=
                        INTEGER:val
                        {: RESULT=new Operation("n", val);:}
                        |DECIMAL:val
                        {: RESULT=new Operation("n", val);:}
                        |LPAREN operation:op RPAREN
                        {: RESULT=op;:}
                        | MINUS:sm factor:op
                        {:
                            Execute.getExecute().addMathSymbol(smleft, smright, MathSymbol.RESTA);
                            RESULT=new Operation("minus", op, null);
                        :}
                        ;

cuerpo_pie::=
                        statement_pie:att fin cuerpo_pie:arr
                        {:
                            arr.add(att);
                            RESULT = arr;
                        :}
                        | statement_pie:att fin
                        {:
                            ArrayList<Attribute> attributes = new ArrayList<>();
                            attributes.add(att);
                            RESULT = attributes;
                        :}
                        ;
statement_pie::=
                        titulo:ti
                        {:
                            RESULT = ti;
                        :}
                        |etiquetas:et
                        {:
                            RESULT = et;
                        :}
                        |valores:va
                        {:
                            RESULT = va;
                        :}
                        |unir:u
                        {:
                            RESULT = u;
                        :}
                        |tipo:tp
                        {:
                            RESULT = tp;
                        :}
                        |total:to
                        {:
                            RESULT = to;
```

```
                        :}
                        |extra:ex
                        {:
                            RESULT = ex;
                        :}
                        ;
etiquetas::=
                        ETIQUETAS COLON array_strings:arr
                        {:
                            Attribute<ArrayNode<String>> att = new Attribute<>(arr, Attribute.ETIQUETAS, arrleft);
                            RESULT = att;
                        :}
                        ;
valores::=
                        VALORES COLON array_operations:arr
                        {:
                            Attribute<ArrayNode<Operation>> att = new Attribute<>(arr, Attribute.VALORES, arrleft);
                            RESULT = att;
                        :}
                        ;
tipo::=
                        TIPO COLON pie_types:st
                        {:
                            Attribute<String> att = new Attribute<>(st, Attribute.TIPO, stleft);
                            RESULT = att;
                        :}
                        ;
pie_types::=
                        CANTIDAD:st
                        {:
                            RESULT = st;
                        :}
                        |PORCENTAJE:st
                        {:
                            RESULT = st;
                        :}
                        ;
total::=
                        TOTAL COLON operation:op
                        {:
                            Attribute<Operation> att = new Attribute<>(op, Attribute.TOTAL, opleft);
                            RESULT = att;
                        :}
                        ;
extra::=
                        EXTRA COLON STRING:st
                        {:
                            st = st.replaceAll("\"", "");
                            Attribute<String> att = new Attribute<>(st, Attribute.EXTRA, stleft);
                            RESULT = att;
                        :}
                        ;
execute::=
                        EJECUTAR LPAREN STRING:st RPAREN fin
                        {:
                            st = st.replaceAll("\"", "");
                            RESULT = st;
                        :}
                        ;
```