

**INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**Manual Tecnico**

**MATERIA:** Compiladores

**ALUMNO:**

Nava Romo Edgar Adrian

Vilches Segundo Galilea Yanely

**PROFESOR:** Tecla Parra Roberto

ÍNDICE

[**1. Objetivo del Documento**](#_ier2xuhsm7ky) **3**

[**2. Manual de Tecnico.**](#_5tjx0zhj9ge4) **3**

[2.1 Gramática.](#_t1ox8q4degq6) 3

[2.2 Diagrama de Clases.](#_a47z744d0i9r) 13

[2.3 Dibujos](#_hywy1tcwqmvd) 14

[2.3.1 Estrella de David](#_jmuyz42kihuj) 14

[2.3.2 Espiral Cuadrada](#_ionxqdpd2pb5) 15

[2.3.3 Ciclos-Espiral](#_dpwnzt6v82iu) 15

[2.3.4 Procedimientos](#_jylwq4p8q5ce) 16

[2.3.5 Flor](#_yd4wy44p955) 16

# 1. Objetivo del Documento

El presente documento tiene como finalidad mostrar al usuario los requerimientos técnicos para que funcione el sistema

# 2. Manual de Tecnico.

## 2.1 Gramática.

%{

import java.lang.Math;

import java.io.\*;

import java.util.StringTokenizer;

import modelo.Configuracion;

%}

%token IF

%token ELSE

%token WHILE

%token FOR

%token COMP

%token DIFERENTES

%token MAYOR

%token MENOR

%token MAYORI

%token MENORI

%token FNCT

%token NUMBER

%token VAR

%token AND

%token OR

%token FUNC

%token RETURN

%token PARAMETRO

%token PROC

%right '='

%left '+' '-'

%left '\*'

%left ';'

%left COMP

%left DIFERENTES

%left MAYOR

%left MAYORI

%left MENOR

%left MENORI

%left '!'

%right RETURN

%%

list:

| list'\n'

| list linea '\n'

;

linea: exp ';' {$$ = $1;}

|stmt {$$ = $1;}

|linea exp ';' {$$ = $1;}

|linea stmt {$$ = $1;}

;

exp: VAR {$$ = new ParserVal(maquina.agregarOperacion("eval")); maquina.agregar($1.sval);}

|'-' exp {$$ = new ParserVal(maquina.agregarOperacion("negativo"));}

|NUMBER {$$ = new ParserVal(maquina.agregarOperacion("constPush"));maquina.agregar($1.dval);}

| VAR '=' exp {$$ = new ParserVal($3.ival);maquina.agregarOperacion("varPush");maquina.agregar($1.sval);maquina.agregarOperacion("asignar");maquina.agregarOperacion("eval");

maquina.agregar($1.sval);

}

| exp '\*' exp {$$ = new ParserVal($1.ival);maquina.agregarOperacion("multiplicar");}

| exp '+' exp {$$ = new ParserVal($1.ival);maquina.agregarOperacion("sumar");}

| exp '-' exp {$$ = new ParserVal($1.ival);maquina.agregarOperacion("restar");}

|'(' exp ')' {$$ = new ParserVal($2.ival);}

| exp COMP exp {maquina.agregarOperacion("comparar");$$ = $1;}

| exp DIFERENTES exp {maquina.agregarOperacion("compararNot");$$ = $1;}

| exp MENOR exp {maquina.agregarOperacion("menor");$$ = $1;}

| exp MENORI exp {maquina.agregarOperacion("menorIgual");$$ = $1;}

| exp MAYOR exp {maquina.agregarOperacion("mayor");$$ = $1;}

| exp MAYORI exp {maquina.agregarOperacion("mayorIgual");$$ = $1;}

| '!' exp {maquina.agregarOperacion("negar");$$ = $2;}

| RETURN exp { $$ = $2; maquina.agregarOperacion("\_return"); }

| PARAMETRO { $$ = new ParserVal(maquina.agregarOperacion("push\_parametro")); maquina.agregar((*int*)$1.ival); }

|nombreProc '(' arglist ')' { $$ = new ParserVal(maquina.agregarOperacionEn("invocar",($1.ival))); maquina.agregar(null); } //instrucciones tiene la estructura necesaria para la lista de argumentos

;

arglist:

|exp {$$ = $1; maquina.agregar("Limite");}

|arglist ',' exp {$$ = $1; maquina.agregar("Limite");}

;

nop: {$$ = new ParserVal(maquina.agregarOperacion("nop"));}

;

stmt:if '(' exp stop ')' '{' linea stop '}' ELSE '{' linea stop'}' {

$$ = $1;maquina.agregar($7.ival, $1.ival + 1);maquina.agregar($12.ival, $1.ival + 2);

maquina.agregar(maquina.numeroDeElementos() - 1, $1.ival + 3);

}

| if '(' exp stop ')' '{' linea stop '}' nop stop{

$$ = $1;maquina.agregar($7.ival, $1.ival + 1);maquina.agregar($10.ival, $1.ival + 2);

maquina.agregar(maquina.numeroDeElementos() - 1, $1.ival + 3);

}

| while '(' exp stop ')' '{' linea stop '}' stop{

$$ = $1;maquina.agregar($7.ival, $1.ival + 1);maquina.agregar($10.ival, $1.ival + 2);

}

| for '(' instrucciones stop ';' exp stop ';' instrucciones stop ')' '{' linea stop '}' stop{

$$ = $1;maquina.agregar($6.ival, $1.ival + 1);maquina.agregar($9.ival, $1.ival + 2);maquina.agregar($13.ival, $1.ival + 3);

maquina.agregar($16.ival, $1.ival + 4);

}

| funcion nombreProc '(' ')' '{' linea '}'

| procedimiento nombreProc '(' ')' '{' linea '}' // definicion de funciones linea es el cuerpo de la funcion

| instruccion '[' arglist ']' ';' {

$$ = new ParserVal($1.ival);maquina.agregar(null);

}

;

instruccion: FNCT {

$$ = new ParserVal(maquina.agregar((Funcion)($1.obj)));

}

;

procedimiento: PROC { maquina.agregarOperacion("declaracion"); }

;

funcion: FUNC { maquina.agregarOperacion("declaracion"); }

;

nombreProc: VAR {$$ = new ParserVal(maquina.agregar($1.sval));}

;

null: {maquina.agregar(null);}

;

stop: {$$ = new ParserVal(maquina.agregarOperacion("stop"));}

;

if: IF {

$$ = new ParserVal(maquina.agregarOperacion("\_if\_then\_else"));maquina.agregarOperacion("stop");maquina.agregarOperacion("stop");

maquina.agregarOperacion("stop");

}

;

while: WHILE {

$$ = new ParserVal(maquina.agregarOperacion("\_while"));maquina.agregarOperacion("stop");

maquina.agregarOperacion("stop");

}

;

for : FOR {

$$ = new ParserVal(maquina.agregarOperacion("\_for"));maquina.agregarOperacion("stop");maquina.agregarOperacion("stop");

maquina.agregarOperacion("stop");maquina.agregarOperacion("stop");

}

instrucciones: { $$ = new ParserVal(maquina.agregarOperacion("nop"));}

|exp {$$ = $1;}

|instrucciones ',' exp {$$ = $1;}

;

%%

TablaDeSimbolos tablaDeSimbolos = new TablaDeSimbolos();

MaquinaDePila maquina = new MaquinaDePila(tablaDeSimbolos);

*int* i = 0;

*int* j = 0;

*double*[][] auxiliar;

Funcion funcionAux;

boolean huboError;

String ins;

StringTokenizer st;

*void* yyerror(String s){

huboError = true;

System.out.println("error:"+s);

System.exit(0);

}

boolean newline;

*int* yylex(){

String s;

*int* tok = 0;

Double d;

if (!st.hasMoreTokens()){

if (!newline){

newline=true;

return '\n';

}

else

return 0;

}

s = st.nextToken();

try{

d = Double.valueOf(s);/\*this MAYOR fail\*/

yylval = new ParserVal(d.doubleValue()); //SEE BELOW

return NUMBER;

}

catch (Exception e){}

if(esVariable(s)){

switch(s){

case "procedure":

return PROC;

case "return":

return RETURN;

case "function":

return FUNC;

case "if":

return IF;

case "else":

return ELSE;

case "while":

return WHILE;

case "for":

return FOR;

default:

break;

}

if(s.charAt(0) == '$'){

yylval = new ParserVal((*int*)Integer.parseInt(s.substring(1)));

return PARAMETRO;

}

boolean esFuncion = false;

Object objeto = tablaDeSimbolos.encontrar(s);

if(objeto instanceof Funcion){

funcionAux = (Funcion)objeto;

yylval = new ParserVal(objeto);

esFuncion = true;

return FNCT;

}

if(!esFuncion){

yylval = new ParserVal(s);

return VAR;

}

}

else{

switch(s){

case "==":

return COMP;

case "!=":

return DIFERENTES;

case "<":

return MENOR;

case "<=":

return MENORI;

case ">":

return MAYOR;

case ">=":

return MAYORI;

default:

break;

}

tok = s.charAt(0);

}

return tok;

}

String reservados[] = {">=", "&&", "||", "<=","==", "!=", "=", "{", "}", ",", "\*", "+", "-", "(", ")", "|", "[", "]", ";", "!", "<", ">"};

public String ajustarCadena(String cadena){

String nueva = "";

boolean encontrado = false;

for(*int* i = 0; i < cadena.length() - 1; i++){

encontrado = false;

for(*int* j = 0; j < reservados.length; j++){

if(cadena.substring(i, i + reservados[j].length()).equals(reservados[j])){

i += reservados[j].length()-1;

nueva += " " + reservados[j] + " ";

encontrado = true;

break;

}

}

if(!encontrado)

nueva += cadena.charAt(i);

}

nueva += cadena.charAt(cadena.length()-1);

return nueva;

}

boolean esVariable(String s){

boolean cumple = true;

for(*int* i = 0; i < reservados.length; i++)

if(s.equals(reservados[i]))

cumple = false;

return cumple;

}

public *void* insertarInstrucciones(){

tablaDeSimbolos.insertar("Rotate", new MaquinaDePila.Girar());

tablaDeSimbolos.insertar("Move", new MaquinaDePila.Avanzar());

}

public Configuracion ejecutarCodigo(String codigo){

st = new StringTokenizer(ajustarCadena(codigo));

newline=false;

yyparse();

if(!huboError)

maquina.ejecutar();

return maquina.getConfiguracion();

}

public boolean compilar(String codigo){

st = new StringTokenizer(ajustarCadena(codigo));

newline=false;

yyparse();

return !huboError;

}

public boolean ejecutarSiguiente(){

return maquina.ejecutarSiguiente();

}

public Configuracion getConfiguracion(){

return maquina.getConfiguracion();

}

public *void* limpiar(){

tablaDeSimbolos = new TablaDeSimbolos();

insertarInstrucciones();

maquina = new MaquinaDePila(tablaDeSimbolos);

}

public Configuracion ejecutar(){

maquina.ejecutar();

return maquina.getConfiguracion();

}

*void* dotest() throws Exception{

insertarInstrucciones();

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

while (true){

huboError = false;

try{

ins = ajustarCadena(in.readLine());

}

catch (Exception e){}

st = new StringTokenizer(ins);

newline=false;

yyparse();

if(!huboError)

maquina.ejecutar();

}

}

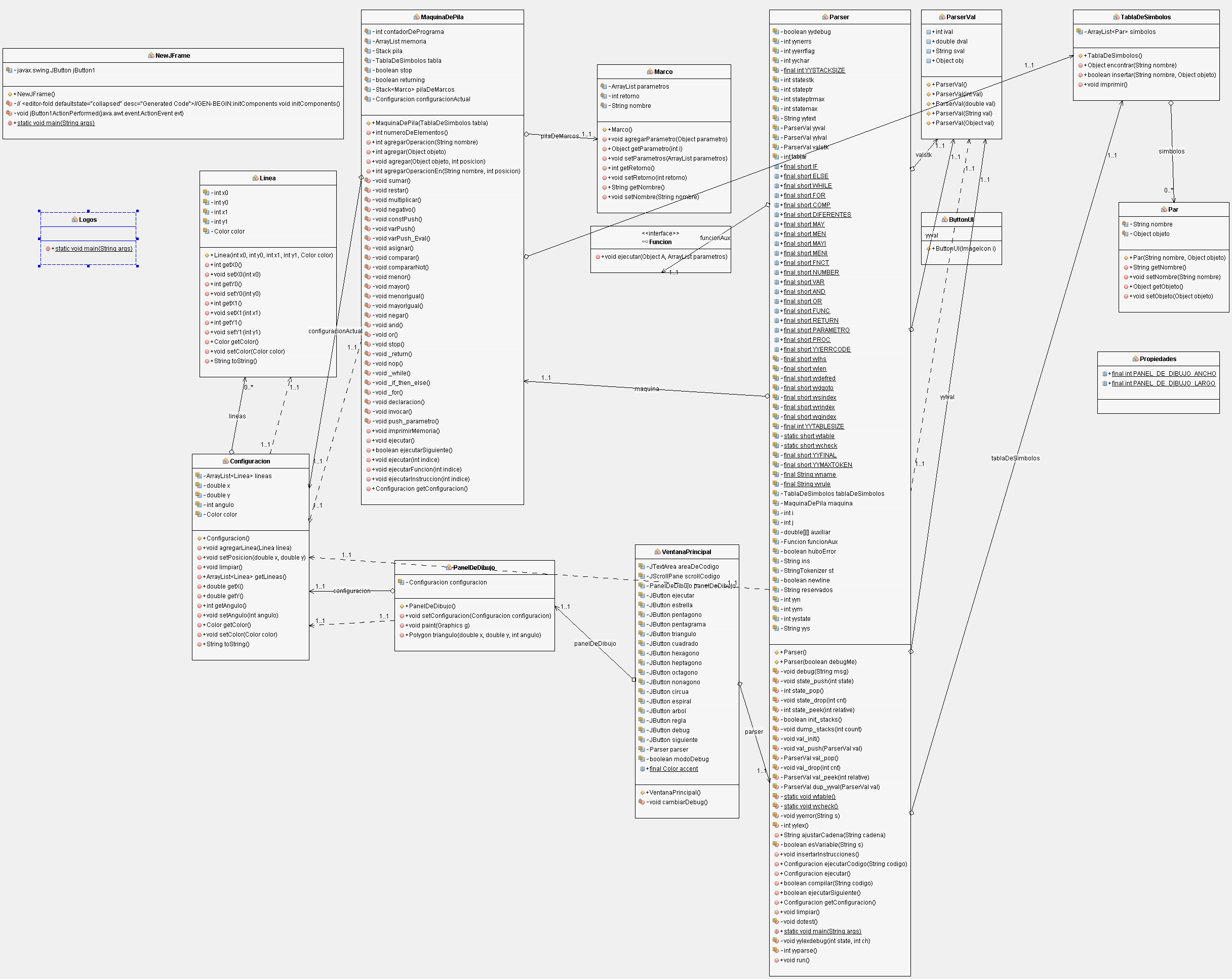
public static *void* main(String args[]) throws Exception{

Parser par = new Parser(false);

par.dotest();

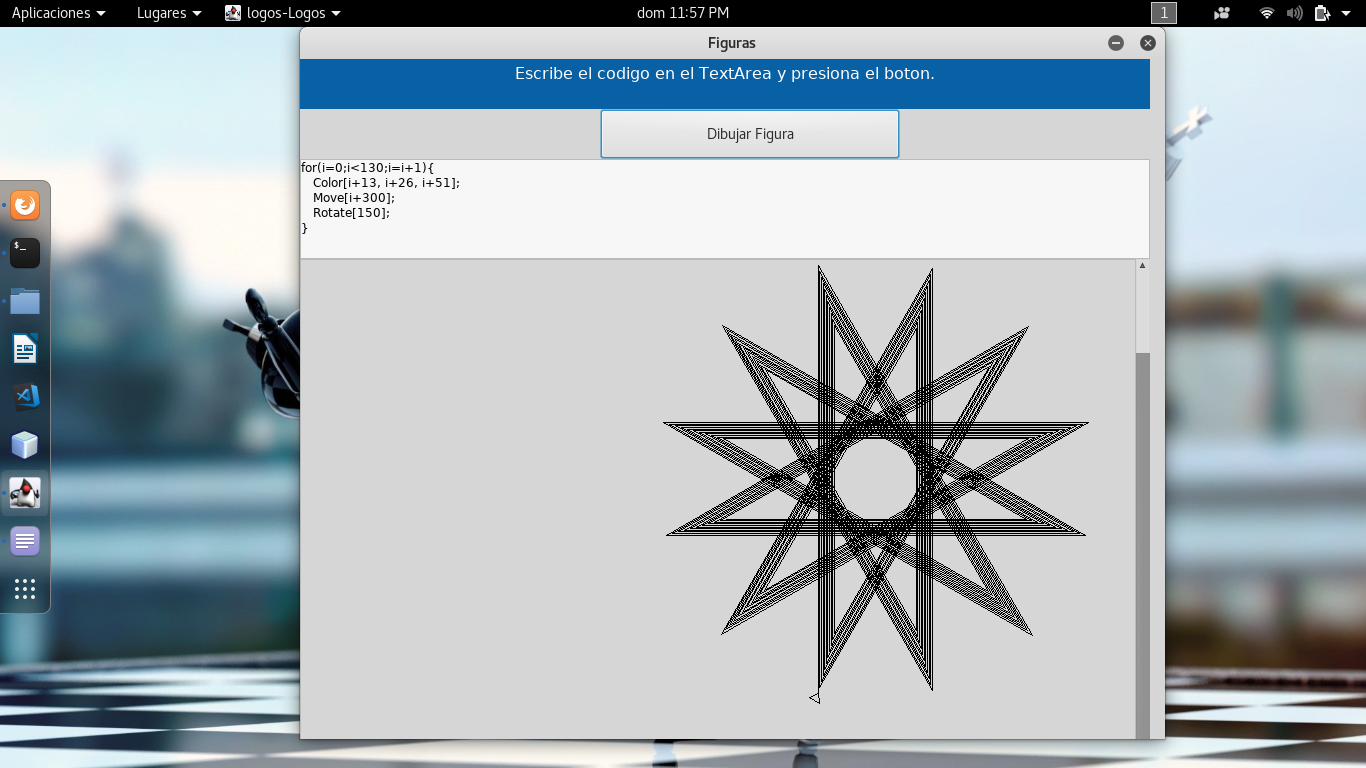
}

## 2.2 Diagrama de Clases.

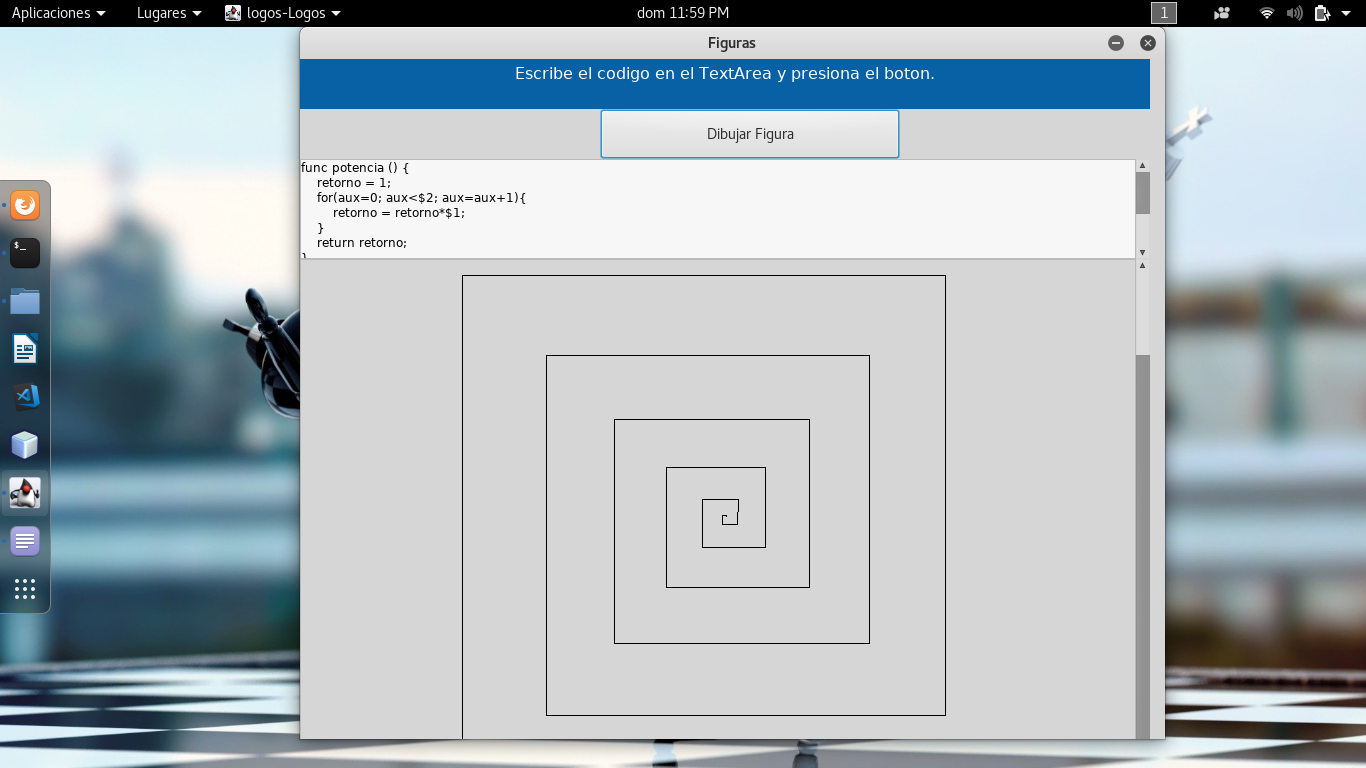


## 2.3 Dibujos

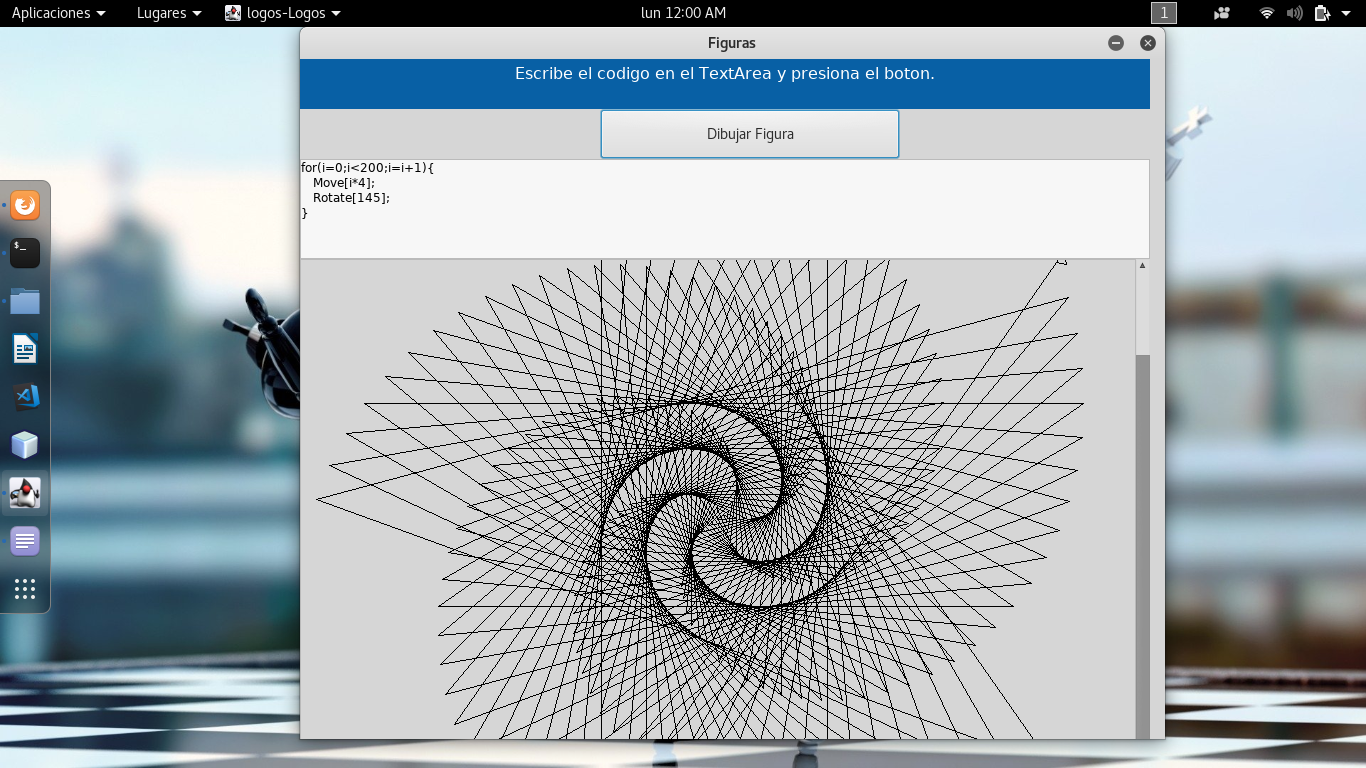
### 2.3.1 Estrella de David



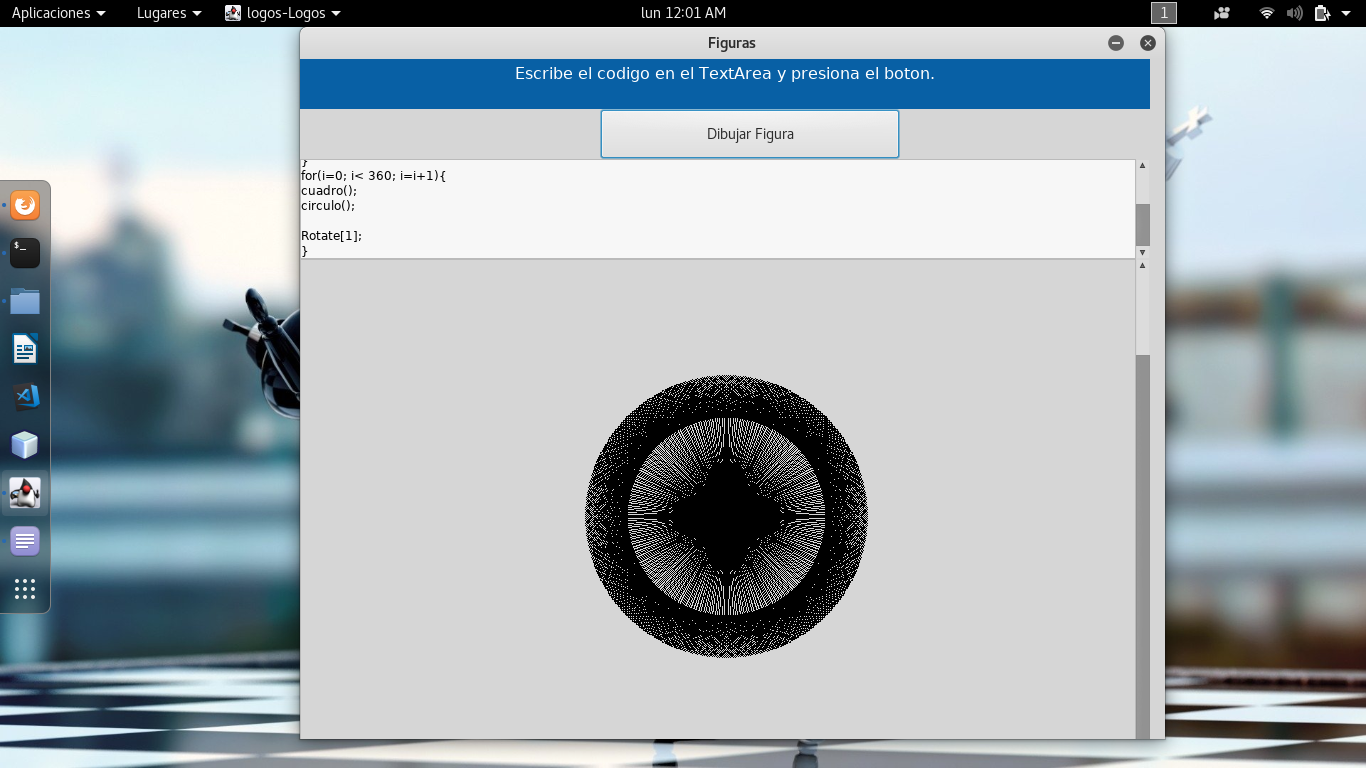
### 2.3.2 Espiral Cuadrada



### 2.3.3 Ciclos-Espiral



### 2.3.4 Procedimientos



### 2.3.5 Flor

