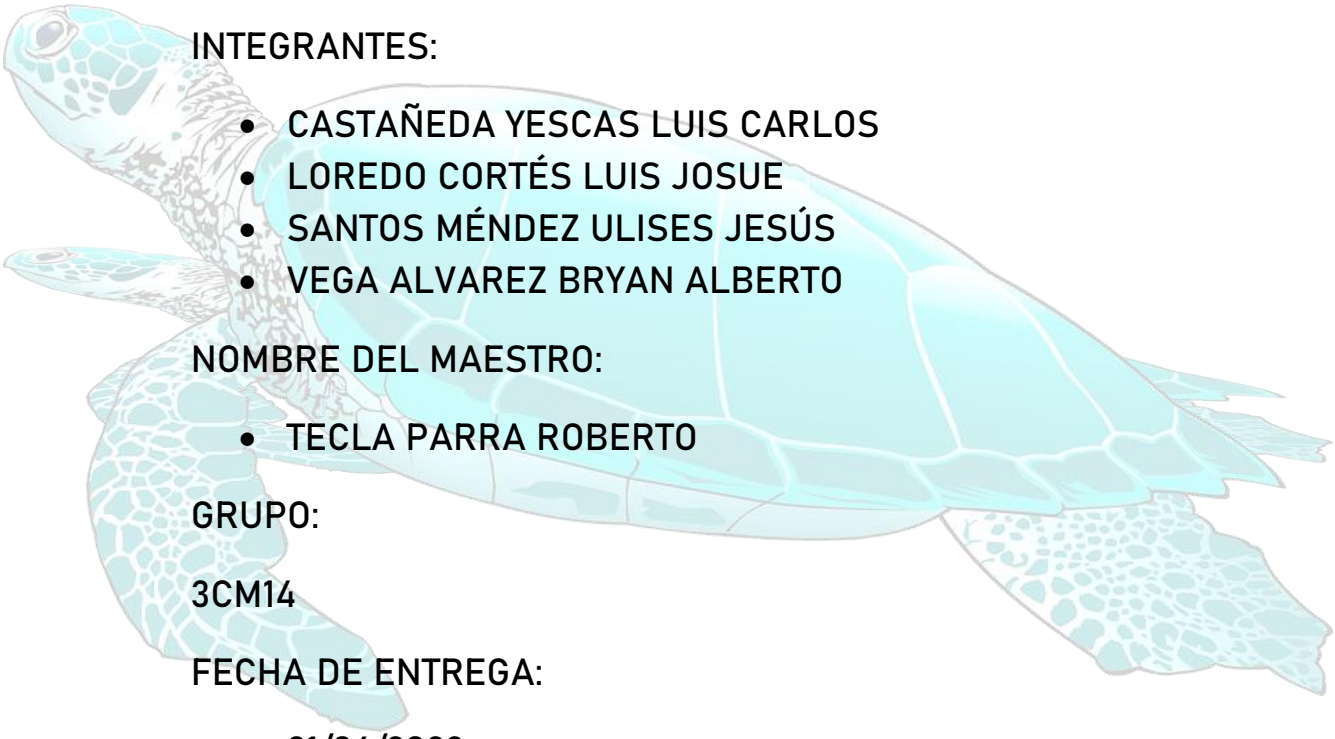
INSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO (ESCOM)

COMPILADORES

PROYECTO:

- DEMO “LOGOS” MANUAL TÉCNICO



INTEGRANTES:

- CASTAÑEDA YESCAS LUIS CARLOS
- LOREDO CORTÉS LUIS JOSUE
- SANTOS MÉNDEZ ULISES JESÚS
- VEGA ALVAREZ BRYAN ALBERTO

NOMBRE DEL MAESTRO:

- TECLA PARRA ROBERTO

GRUPO:

3CM14

FECHA DE ENTREGA:

- 21/06/2023

Manual Técnico Demo “Logos”

INTRODUCCIÓN:

El lenguaje de programación Logo, desarrollado en la década de 1960 en el Instituto de Tecnología de Massachusetts (MIT), se diseñó específicamente para enseñar a programar a niños y jóvenes. Fue creado por Seymour Papert, Wally Feurzeig y Cynthia Solomon, quienes buscaban desarrollar un entorno de programación accesible y amigable para los niños.

El enfoque principal de Logo era brindar a los niños una herramienta para explorar conceptos de programación y desarrollar habilidades de pensamiento computacional a través de la resolución de problemas prácticos. Logo utilizaba una interfaz gráfica en la que los niños podían dar instrucciones a una "tortuga" virtual que se movía en la pantalla y dibujaba formas.

La idea central de Logo era fomentar la creatividad y la experimentación. Los niños podían programar la tortuga para que dibujara figuras geométricas, patrones y diseños, lo que les permitía visualizar y comprender los conceptos de programación, como el control de flujo, la iteración y las estructuras de datos.

Logo se basaba en un lenguaje de programación de alto nivel con una sintaxis sencilla y comprensible. Los comandos eran expresados en forma de palabras y frases en inglés, lo que facilitaba su comprensión y uso para los niños.

Además del aspecto gráfico, Logo también permitía a los niños escribir programas utilizando instrucciones de texto. Esto les brindaba una mayor flexibilidad y capacidad de expresión en la programación.

Logo se convirtió en una herramienta popular en las escuelas y se utilizó para enseñar una variedad de conceptos, desde lógica y matemáticas hasta resolución de problemas y pensamiento algorítmico. Su enfoque en la experimentación y la creatividad ayudó a los niños a desarrollar habilidades cognitivas y de razonamiento, así como a fomentar su interés en la informática y la programación.

El desarrollo del presente proyecto está enfocado en que sea posible enseñar a programar a personas con pocos conocimientos de una forma divertida, esto será con ayuda de una interfaz gráfica amigable que le permita ver las instrucciones, así como meter cualquier comando que sea válido y que le permita dibujar cualquier figura que se le pueda ocurrir.

OBJETIVO:

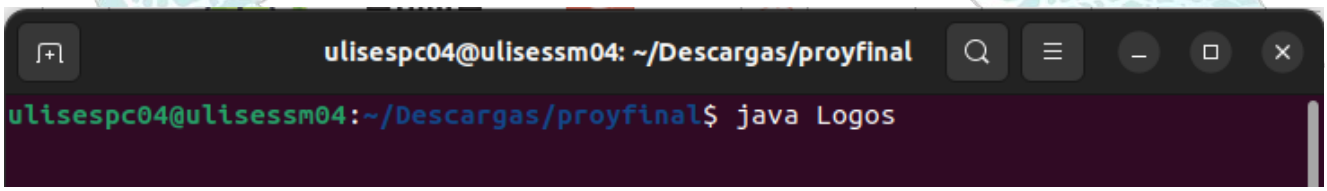
- El objetivo de este trabajo es dar referencia técnica acerca del proyecto para que pueda ser entendido por alguna persona con los conocimientos técnicos aptos y así poder ser modificado o incluso optimizado.
- Tener una GUI que permita al usuario manejarla de forma intuitiva y fácil, lo suficiente para aprender de una forma rápida.

REQUERIMIENTOS TÉCNICOS:

- Java.
- Java Development Kit (JDK).
- Java Runtime Edition (JRE).
- Byacc.
- 4 GB RAM.
- Disco duro de cualquier capacidad.
- Windows/Linux

EJECUCIÓN:

Lo primero que se hará será abrir una terminal en Linux donde se encuentren los archivos correspondientes al proyecto, una vez que se ha abierto la terminal se ingresa el comando Java Logos, solo se deberá de ingresar un comando por el mismo motivo se les esta entregando una carpeta ya compilada aún así si hubiera algún cambio a futuro basta con ejecutar Javac *.java para obtener las clases y finalmente ejecutamos Java Logos



Ahora que está en ejecución el programa, y principalmente consta de un botón para consultar las instrucciones, un área para pegar o ingresar tu propio código, un botón para realizar la inserción de un archivo .txt donde venga tu código, un botón de Inicio, Borrar, Debug y cuando se active el debug te dejará verlo paso a paso con el botón de avance.

Una vez que se ejecuta el programa tendremos la interfaz gráfica mostrada en la Imagen 5, en donde

tenemos los siguientes componentes:

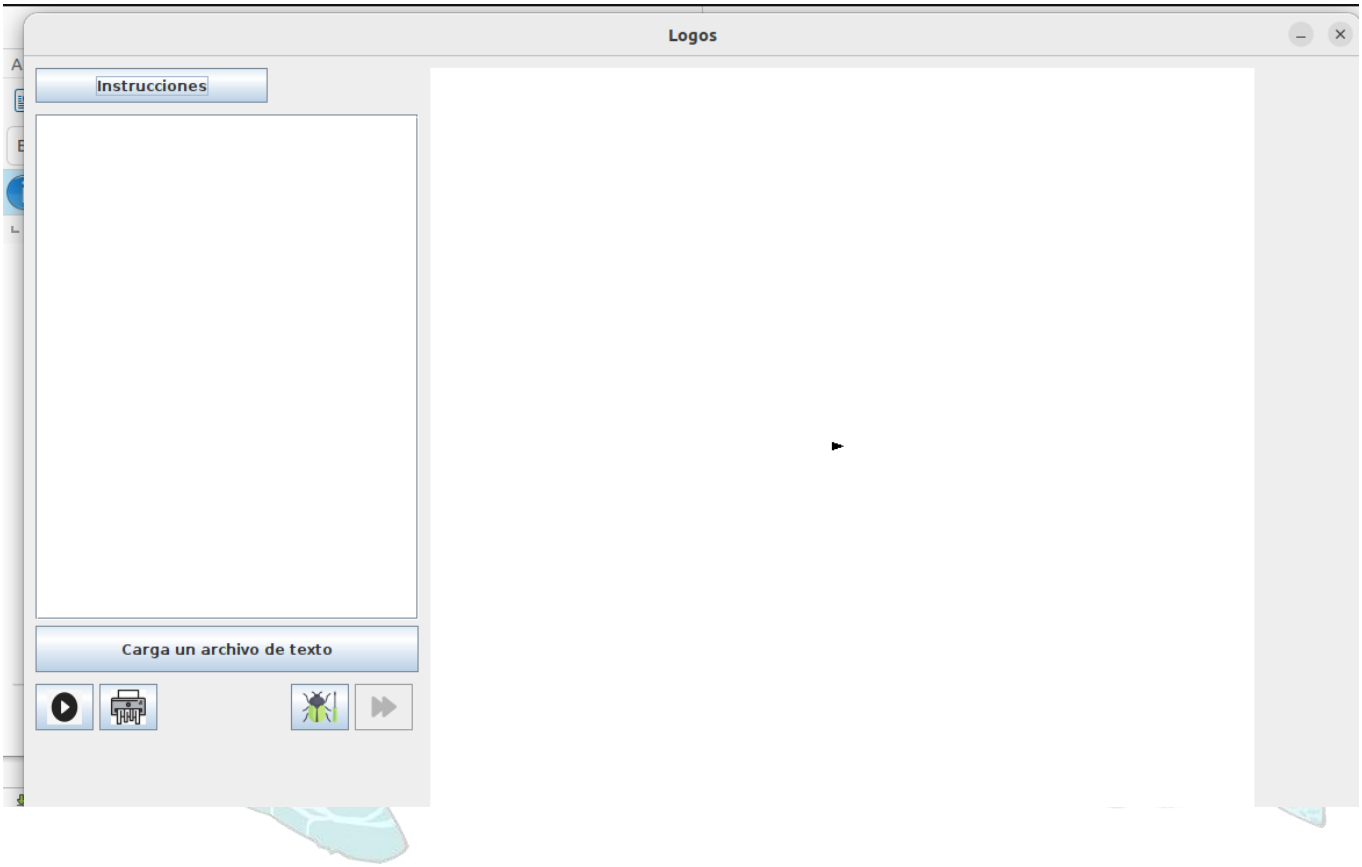
• Un área de texto, donde se ingresan los comandos del lenguaje.

• Un botón que permite dibujar las acciones descritas por los comandos.

• Un botón que permite borrar lo que esta dibujado en el panel y nos permite realizar el botón del siguiente punto, si fue seleccionado anteriormente y se le vuelve a seleccionar regresamos se nos deshabilita la opción del botón siguiente y guarda el estado en el panel lo que hicimos mientras lo seleccionamos por primera vez.

• Un botón que dibuja paso a paso la figura.

• Un panel, donde se visualiza el dibujo.



Expresiones regulares:

Las siguientes son expresiones regulares:

• FORWARD[double];

• TURN[double];

• COLOR[double, double, double];

• PenUP[];

• PenDOWN[];

• for(exploración inicial; condición; expresión final){}

• while(condición){ declaracion(es)}

• if (condición){instrucciones}else{instrucciones} proc [nombre](){Bloque de instrucciones} **Para usar parámetros usar $n, donde n es el parámetro para usar**, en las llamadas los parámetros se separan con una ‘,’.

• func [nombre](){ Bloque de instrucciones return valor; } **Para usar parámetros usar $n, donde n es el parámetro para usar**, en las llamadas los parámetros se separan con una ‘,’.

Condiciones disponibles:

• ==, comparación, entre enteros

• !=, diferente, entre enteros

• <=, operación menor o igual, entre enteros.

• >, operación mayor, entre enteros.

• >=, operación mayor o igual, entre enteros.

• &&, operación “and”, entre enteros. ||, operación “or”, entre enteros.

GRAMÁTICA:

```
%token IF
%token ELSE
%token WHILE
%token FOR
%token COMP
%token DIFERENTES
%token MAY
%token MEN
%token MAYI
%token MENI
%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 MAY
%left MAYI
%left MEN
%left MENI
%left '!'
%left AND
%left OR
%right RETURN
%%
```

```
list:
        | list'\n'
        | list linea '\n'
        ;

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

    exp:  VAR {
                $$ = new ParserVal(maquina.agregarOperacion("varPush_Eval"));
                maquina.agregar($1.sval);
            }
        |'-' exp {
                $$ = 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("varPush_Eval");
                maquina.agregar($1.sval);
            }
        | exp '*' exp {
                $$ = new ParserVal($1.ival);
                maquina.agregarOperacion("MUL");
```

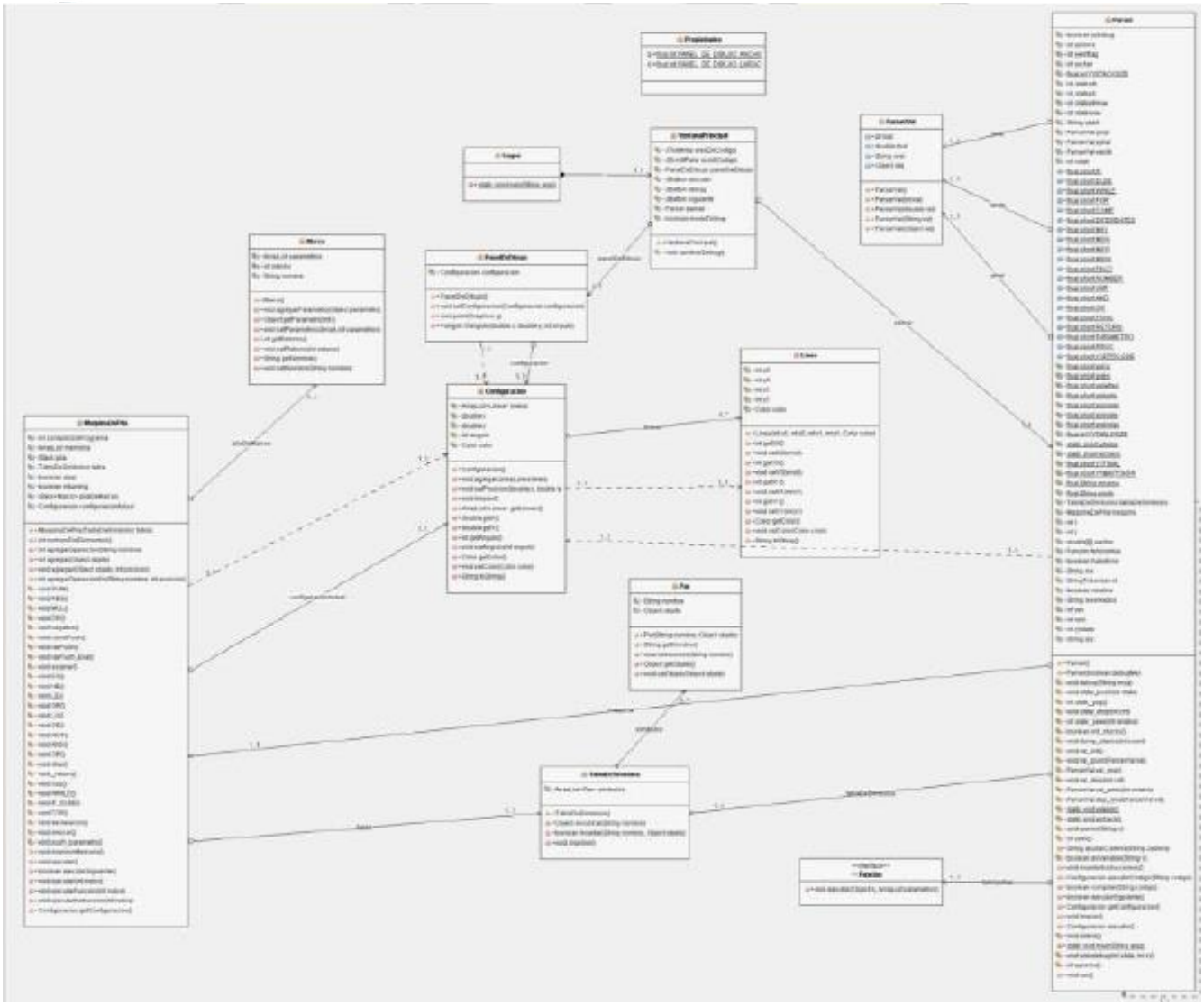
```
            }
                | exp '/' exp {
                $$ = new ParserVal($1.ival);
                maquina.agregarOperacion("DIV");
            }
        | exp '+' exp {
                $$ = new ParserVal($1.ival);
                maquina.agregarOperacion("SUM");
            }
        | exp '-' exp {
                $$ = new ParserVal($1.ival);
                maquina.agregarOperacion("RES");
            }
        |'(' exp ')' {
                $$ = new ParserVal($2.ival);
            }
        | exp COMP exp {
                 maquina.agregarOperacion("EQ");
                 $$ = $1;
            }
        | exp DIFERENTES exp {
                 maquina.agregarOperacion("NE");
                 $$ = $1;
            }
        | exp MEN exp {
                 maquina.agregarOperacion("LE");
                 $$ = $1;
            }
        | exp MENI exp {
                 maquina.agregarOperacion("LQ");
                 $$ = $1;
            }
        | exp MAY exp {
                 maquina.agregarOperacion("GR");
                 $$ = $1;
            }
        | exp MAYI exp {
                 maquina.agregarOperacion("GE");
                 $$ = $1;
            }
        | exp AND exp {
                maquina.agregarOperacion("AND");
                 $$ = $1;
            }
        | exp OR exp {
                maquina.agregarOperacion("OR");
                 $$ = $1;
```

```
            }
        | '!' exp {
                maquina.agregarOperacion("NOT");
                $$ = $2;
            }
        | RETURN exp { $$ = $2; maquina.agregarOperacion("_return"); }

        | PARAMETRO { $$ = new ParserVal(maquina.agregarOperacion("push_parame-
tro")); maquina.agregar((int)$1.ival); }

        |nombreProc '(' arglist ')' { $$ = new ParserVal(maquina.agregarOperacio-
nEn("invocar",($1.ival))); maquina.agregar(null); } //instrucciones tiene la estruc-
tura 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 null '}'
        | procedimiento nombreProc '(' ')' '{' linea null '}'
        | 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_ELSE"));
            maquina.agregarOperacion("stop");//then
            maquina.agregarOperacion("stop");//else
            maquina.agregarOperacion("stop");//siguiente comando
        }
        ;

    while: WHILE {
            $$ = new ParserVal(maquina.agregarOperacion("WHILE"));
            maquina.agregarOperacion("stop");//cuerpo
            maquina.agregarOperacion("stop");//final
        }
        ;

    for : FOR {
            $$ = new ParserVal(maquina.agregarOperacion("FOR"));
            maquina.agregarOperacion("stop");//condicion
            maquina.agregarOperacion("stop");//instrucción final
            maquina.agregarOperacion("stop");//cuerpo
```

```
            maquina.agregarOperacion("stop");//final
        }

    instrucciones: { $$ = new ParserVal(maquina.agregarOperacion("nop"));}
        |exp {$$ = $1;}
        |instrucciones ',' exp {$$ = $1;}
        ;


%%
```

DIAGRAMA DE CLASES



EXPLICACIÓN DE ALTO NIVEL

Para la realización de este proyecto fue necesario hacer uso de todos los conocimientos aprendidos en esta materia, para ello lo primero es hacer nuestra clase Par la cual es como las que hemos usado en anteriores ocasiones para las prácticas en este caso tendremos un nombre y un objeto, en segunda necesitamos nuestra tabla de símbolos la cual contendrá todos nuestras instrucciones las cuales son: TURN, FORWARD, COLOR, PenUP y PenDOWN, después necesitamos nuestra máquina de pila, la cual se encuentra en el archivo Java que posee el mismo nombre que es sin duda alguna una de las partes más esenciales de nuestro proyecto ya que es ahí donde sucede todo ya que tiene el funcionamiento de las condiciones, de las operaciones que podemos hacer y por supuesto de nuestras instrucciones el código es el siguiente:

```
public static class Girar implements Funcion{
        @Override
        public void ejecutar(Object A, ArrayList parametros) {
            Configuracion configuracion = (Configuracion)A;
            int angulo = (configuracion.getAngulo() + (int)(double)parame-
tros.get(0))%360;
            configuracion.setAngulo(angulo);
        }
    }

    public static class Avanzar implements Funcion{
        @Override
        public void ejecutar(Object A, ArrayList parametros) {
            Configuracion configuracion = (Configuracion)A;
            int angulo = configuracion.getAngulo();
            double x0 = configuracion.getX();
            double y0 = configuracion.getY();
            double x1 = x0 + Math.cos(Math.toRadians(angulo))*(double)parame-
tros.get(0);
            double y1 = y0 + Math.sin(Math.toRadians(angulo))*(double)parame-
tros.get(0);
            configuracion.setPosicion(x1, y1);
            configuracion.agregarLinea(new Linea((int)x0,(int)y0,(int)x1,(int)y1,
configuracion.getColor()));
        }
    }

    public static class CambiarColor implements Funcion{
        @Override
        public void ejecutar(Object A, ArrayList parametros) {
            Configuracion configuracion = (Configuracion)A;
            configuracion.setColor(new Color((int)(double)parametros.get(0)%256,
(int)(double)parametros.get(1)%256, (int)(double)parametros.get(2)%256));
```

```
        }
    }

    public static class SubirPincel implements Funcion{
        @Override
        public void ejecutar(Object A, ArrayList parametros) {
            Configuracion configuracion = (Configuracion)A;
            configuracion.setColor(Color.GRAY);
        }
    }
    public static class BajarPincel implements Funcion{
        @Override
        public void ejecutar(Object A, ArrayList parametros) {
            Configuracion configuracion = (Configuracion)A;
            configuracion.setColor(Color.BLACK);
        }
    }
}
```

En estos fragmentos de código encontramos los más importantes de nuestro proyecto y es que aquí creemos y decimos el comportamiento que este tendrá en la ejecución del proyecto, en caso de querer añadir más funcionalidades al proyecto es qui en donde tendremos que remitirnos y evidentemente al archivo P2.y ya que necesitamos reflejar los cambios en Parser y ParserVal que se generan al compilar el archivo de YACC, recordemos también que hacemos uso de Marcos. Todo lo anterior es la funcionalidad de Java, pero esto no se podría hacer sin nuestro archivo YACC el cual ya hemos visto, al menos la gramática, aquí solamente llamamos a las funciones creadas en nuestra máquina de pila, asignamos nuestros caracteres reservados a nuestras funciones creadas con anterioridad todo esto en nuestro yylex y finalmente añadimos funcionalidad a nuestros botones en la GUI.

Por ultimo y no mucho menos importante nuestra GUI, en donde nos encontramos con el archivo Configuración que no es más que nada el que nos permite guardar las líneas que vamos añadiendo mediante un ArrayList y para esto tendremos que usar nuestra Linea.java en donde podemos cambiar el color y definimos la dirección y el tamaño que esta tendrán, se anexa Configuracion.java ya que nos permite la visualización de las líneas que hagamos, pero, como podríamos observar las grandes maravillas que podemos hacer con nuestro proyecto si no tenemos un lienzo en donde poder verla, por lo que tenemos nuestro PanelDeDibujo.java y que ademas es qui en donde dibujamos nuestro triangulo que nos indica la dirección hacia donde se dibujara la siguiente acción que hagamos. Finalmente tenemos nuestra VentanaPrincipal.java y Logos.java los cuales no tiene tanto misterio ya que en una solo ponemos los componentes que nuestro usuario interactuará con ellos y para tener un buen formato de presentación y en el otro solo ejecutamos el programa.