1S-2024

OLC 1 - PROYECTO 1

MANUAL TECNICO

Sebastian Solares
202004822

Manual técnico

Lenguaje utilizado: Java es un lenguaje de programación de alto nivel y orientado a objetos, desarrollado por Sun Microsystems (adquirido posteriormente por Oracle). Fue lanzado por primera vez en 1995 y desde entonces se ha convertido en uno de los lenguajes de programación más populares y ampliamente utilizados en el mundo.

Algunas características importantes de Java incluyen su portabilidad, ya que el código escrito en Java puede ejecutarse en cualquier plataforma que tenga un entorno de tiempo de ejecución Java (JRE) instalado, su robustez y seguridad debido a su sistema de manejo de excepciones y verificación de tipos, y su amplia adopción en la industria, especialmente en el desarrollo de aplicaciones empresariales, aplicaciones móviles (Android), sistemas embebidos y aplicaciones web.

Herramientas utilizadas: Jfree chart, jflex, cup, linkedlist, hashmap

Analizador léxico: para realizar el análisis léxico se utilizo jflex en esta parte se agregaron las reglas léxicas, símbolos etc

```
// ------------  Reglas Lexicas --------------------

//entero = [0-9]+

comentario_oneL = "!"[^\r\n]* (\r|\n|\r\n)?
comentario_multL = \<\!([^"!>"]|[\r|\f|\s|\t|\n])*\!\>


numero = [0-9]+("."[0-9]+)?
id = [a-zA-Z_][a-zA-Z0-9_]*
id_array = @([a-zA-Z_][a-zA-Z0-9_]*)
cadena = [\'][^\']*[\']|[\"][^\"]*[\"]
blancos = [ \t\f\r\n]+


%%
```

```
<YYINITIAL>{
    // ------------  Simbolos ------------------------

    ";"    { System.out.println("Reconocio punto y coma,lexema: "+yytext());

             Token tmp = new Token(yytext(),"delimitador",yyline,yycolumn);
             tokLex.add(tmp);
             return new Symbol(sym.PUNTOYCOMA, yycolumn, yyline, yytext()); }

    ":"    { System.out.println("Reconocio Dos puntos ,lexema: "+yytext());
             Token tmp = new Token(yytext(),"delimitador",yyline,yycolumn);
             tokLex.add(tmp);
             return new Symbol(sym.DOSPUNTOS, yycolumn, yyline, yytext()); }

    ","   { System.out.println("Reconocio coma ,lexema: "+yytext());
             Token tmp = new Token(yytext(),"delimitador",yyline,yycolumn);
             tokLex.add(tmp);
             return new Symbol(sym.COMA, yycolumn, yyline, yytext()); }

    "<-"   { System.out.println("Reconocio Assignacion llave ,lexema: "+yytext());
             Token tmp = new Token(yytext(),"delimitador",yyline,yycolumn);
             tokLex.add(tmp);
             return new Symbol(sym.ASSIGN, yycolumn, yyline, yytext()); }

    "->"   { System.out.println("Reconocio Assignacion llave derecha  ,lexema: "+yytext());
             Token tmp = new Token(yytext(),"delimitador",yyline,yycolumn);
             tokLex.add(tmp);
             return new Symbol(sym.ASSIGN2, yycolumn, yyline, yytext()); }
```

```
// ------------  Palabras ------------------------


"var"           { System.out.println("Reconocio palabra ,lexema: "+yytext());
                  Token tmp = new Token(yytext(),"Palabra Reservada",yyline,yycolumn);
                  tokLex.add(tmp);
                    return new Symbol(sym.VAR, yycolumn, yyline, yytext()); }

"double"        { System.out.println("Reconocio tipo de dato ,lexema: "+yytext());
                  Token tmp = new Token(yytext(),"Palabra Reservada",yyline,yycolumn);
                  tokLex.add(tmp);
                  return new Symbol(sym.DOUBLE, yycolumn, yyline, yytext()); }

"char[]"        { System.out.println("Reconocio tipo de dato ,lexema: "+yytext());
                  Token tmp = new Token(yytext(),"Palabra Reservada",yyline,yycolumn);
                  tokLex.add(tmp);
                  return new Symbol(sym.CHAR, yycolumn, yyline, yytext()); }

"END PROGRAM"   { System.out.println("Reconocio palabra ,lexema: "+yytext());
                  Token tmp = new Token(yytext(),"Palabra Reservada",yyline,yycolumn);
                  tokLex.add(tmp);
                  return new Symbol(sym.ENDPROGRAM, yycolumn, yyline, yytext()); }
```

Analizador sintáctico: para realizar esta parte se utilizó la librería cup de java, en esta parte se declararon los terminales y no terminales como también todas las producciones necesarias para realizar el proyecto.

```
//------> Declaración de terminales
terminal String NUMERO,CADENA,COMA,COR_IZQ,COR_DER,ASSIGN,ASSIGN2,DOSPUNTOS,PUNTOYCOMA;
terminal String ID,ID_ARRAY,CONSOLE,COLUMN,PRINT,IGUAL,PAR_IZQ,PAR_DER;
terminal String sum,res,mul,mod,div,media,mediana,moda,varianza,max,min;
terminal String VAR,DOUBLE,CHAR,ARR,END,PROGRAM,ENDPROGRAM;
terminal String graphbar,graphpie,graphline,histogram;
terminal String titulo,ejex,ejey,titulox,tituloy,values,label,exec;




//------> Declaración de no terminales
non terminal INICIO;
non terminal sentencia INSTRUCCION;
non terminal DECLARACION;
non terminal TIPO;
non terminal OP,OP2,OP3,OP4;
non terminal EXP_TITULO;
```

```
//------> Definir Simbolo Inicial
start with INICIO;


// ------------>  Producciones  <------------

//---------->Gramatica para el inicio<-------------
INICIO ::= PROGRAM INSTRUCCIONES:b ENDPROGRAM {: parser.AST=b; :};


INSTRUCCIONES ::= INSTRUCCIONES:a INSTRUCCION:b {: RESULT= a; RESULT.add(b); :}
                | INSTRUCCION: a                {: RESULT = new LinkedList<>(); RESULT.add(a); :};


INSTRUCCION::=   IMPRIMIR_EXP:a  {:RESULT =a; :}
               |IMPRIMIR_ARRAY:a  {: RESULT=a; :}
               |DECLARACION:a  {: RESULT=new func("declarada variable"); :}
               |Dec_Arreglo:a  {:RESULT=new func("declarada arreglo");:}
               |GRAFBARRAS  {:RESULT= new func("nuevagraficabarras");:}
               |GRAFPIE  {:RESULT= new func("NUEVA GRAFICA PIE");:}
               |GRAFLINEAL  {:RESULT= new func("NUEVA GRAFICA lineal");:}
               |GRAFHISTO  {:RESULT= new func("NUEVA GRAFICA HISTOGRAMA");:}
               |error:e PUNTOYCOMA {:RESULT= new erro("error sint"); :};
```

Funcionalidad del proyecto: para la parte de toda la lógica del proyecto se creo una clase en la cual se realiza toda la parte lógica, en esta clase están todas la s funciones para realizar operaciones, buscar variables etc.

```
public static String buscarVariable(String id) {
    declaracion miDeclaracion = new declaracion("miID", "miValor", 1, 1);

    // Llamada a la función obtenerValorPorID
    String valorEncontrado = miDeclaracion.obtenerValorPorID(id);

    if (valorEncontrado != null) {
        System.out.println("Valor encontrado: " + valorEncontrado);
        return valorEncontrado;
    } else {
        System.out.println("ID no encontrado.");
    }
    return valorEncontrado;

}
```

```
System.out.println("Empezamos con este ID: " + id);
switch (id_min) {
    case "sum":
        resultado = numero1 + numero2;
        System.out.println("sum: " + resultado);
        break;
    case "res":
        resultado = numero1 - numero2;
        System.out.println("res: " + resultado);
        break;
    case "mul":
        resultado = numero1 * numero2;
        System.out.println("mul: " + resultado);
        break;
    case "div":

        if (numero2 != 0) {
            resultado = numero1 / numero2;
            System.out.println("div: " + resultado);

        } else {
            System.out.println("Error: División por cero no permitida.");
        }
        break;
    case "mod":

        if (numero2 != 0) {
            resultado = numero1 % numero2;
            System.out.println("div: " + resultado);

        } else {
            System.out.println("Error: División por cero no permitida.");
        }
        break;
    default:
        System.out.println("Error: Operación no reconocida.");
```

Funcionalidad print: para esta funcionalidad se utilizo el patrón interprete, en el cual me ayudo a escribir en consola.

```
public print(LinkedList<Expresions> exps, int Line, int Column,String titulo){

    this.Column = Column;
    this.Line = Line;
    this.titulo =titulo;
    this.exps = exps;

}

@Override
public Object ejecutar(){

    MiSingleton singleton = MiSingleton.obtenerInstancia();


    singleton.add_consola(titulo);


    for(Expresions exp: this.exps){
        Types run_exp = exp.ejecutarexp();

        if(run_exp != null){
            if(run_exp.getType() == Type.STRING){
                singleton.add_consola(run_exp.getString_val());

            }else{
                singleton.add_consola(Double.toString(run_exp.getDouble_val()));
            }
            singleton.add_consola(", ");
        }
    }

    singleton.add_consola("\n");



    return null;
```

Creación de graficas: para la parte de las graficas se utilizo la herramienta jfreechart que ayuda a crear las graficas del proyecto, los datos se mandan a mi interfaz grafica y en ese punto se maneja todo lo de jfreechart.

```
// Recorrer el HashMap para obtener las claves y los valores
for (Map.Entry<String, String> entry : parser.graficasBarras.entrySet()) {
    String clave = entry.getKey();
    String valor = entry.getValue();

    switch (clave) {
        case "ejeX":
            ejex = valor;

            break;
        case "ejeY":
            ejey = valor;
            break;
        case "tituloX":
            tituloX = valor;
            tituloX = tituloX.replace("\"", "");
            break;
        case "tituloY":
            tituloY = valor;
            tituloY = tituloY.replace("\"", "");

            break;
        case "titulo":
            titulo = valor;
            titulo = titulo.replace("\"", "");
            break;

    }

}

String[] listEjex = ejex.split("[=,]");
String[] listEjey = ejey.split("[=,]");

DefaultCategoryDataset datos = new DefaultCategoryDataset();
```

Para almacenar los valores de las graficas se utilizo hashmap para guardar el clave valor y luego encontrar los valores e implementarla en jfreechart

```
JFreeChart grafica_barras = ChartFactory.createBarChart3D(
            titulo,
            tituloX,
            tituloY,
            datos,
            PlotOrientation.VERTICAL,
            true,
            true,
            false
);
list_graficas.add(grafica_barras);
```

Y cada grafica se guardo en una lista para luego poder recorrerlas con los botones siguiente y anterior.

Reportes: para los reportes de tokens, errores y símbolos se utilizó una clase con getter y setters para cada una para ir guardando los datos y luego llamar dicha clase e irla recorriendo y almacenar los datos en los HTML correspondientes para luego crearlos.

```
 * @author sebastiansolares
 */
public class Token {
    //Atributos
    private String token;

    private String description;
    private int Fila;
    private int Columna;

    //Constructor
    public Token(String token, String description,int Fila, int Columna) {
        this.token = token;
        this.description = description;
        this.Fila =Fila;
        this.Columna=Columna;

    }

    //Getters y Setters
    public String getToken() {
        return token;
    }
    public void setToken(String token) {
        this.token = token;
    }

    public String getDescription() {
        return description;
    }
    public void setDesciption(String description) {
        this.description = description;
    }
```

Creación de HTML: como se dijo anterior mente se leen los valores y se almacenan en el HTML correspondiente

```java
// Crear el estilo del HTML tokens
String htmlstyle3 = "<!DOCTYPE html>"
        + "<html>"
        + "<head>"
        + "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=iso-8859-1\">"
        + "<title>Simbolos</title>"
        + "</head>"
        + "<style>"
        + "table, th {background-color: #D7C0AE;} td { border: 1px solid rgb(31, 31, 31);"
        + "border-collapse: collapse;"
        + "background-color: #D7C0AE;"
        + "}"
        + "th:nth-child(even),td:nth-child(even) {"
        + "background-color: #EEE3CB;"
        + "}"
        + "</style>"
        + "<body bgcolor=\"B7C4CF\">"
        + "<center>";

// Crear el contenido de la tabla
String html3 = htmlstyle3 + "<table border=1><tr><th>No.</th><th>Nombre</th><th>tipo</th><th>valor</th><th>fila</th><th>
int count3 =1;
for (int i = 0; i < parser.decSimbol.size(); i++) {
    html3 += "<tr><td><center>" + count3+ "</center></td><td><center>"
            + parser.decSimbol.get(i).getSimbolos() + "</center></td><td><center>"
            + parser.decSimbol.get(i).getTipo()+ "</center></td><td><center>"
            + parser.decSimbol.get(i).getValor()+ "</center></td><td><center>"
            + parser.decSimbol.get(i).getFila()+ "</center></td><td><center>"
            + parser.decSimbol.get(i).getColumna()+ "</center></td></tr>";

    count3++;
}


html3 += "</table></center></body></html>";

// Guardar en un archivo llamado "errores.html"
File file3 = new File("Reportes/Symbolos.html");
BufferedWriter bw3 = new BufferedWriter(new FileWriter(file3));
bw3.write(html3);
```

Interfaz gráfica: para la interfaz grafica se utilizo drag and drop y en cada función creada por java se realizaron varias cosas, como guardar archivo, cargar archivo etc

```java
    /**
     * Creates new form interfaz
     */
    public interfaz() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code

    private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:

        newArchivo();

    }

    private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        openArchivo();
    }

    private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        guardarArchivo();
    }

    private void jMenuItem4ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        //esta funcion sirve para eliminar la pestana actual
        int selectedIndex = jTabbedPane1.getSelectedIndex();
        if (selectedIndex != -1) {
            jTabbedPane1.remove(selectedIndex);
        }
    }
```