

Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Laboratorio Organización de Lenguajes y Compiladores 1

Proyecto 1: DataForge

Manual Técnico

Nombre: Josué Daniel Rojché García

Carné: 201901103

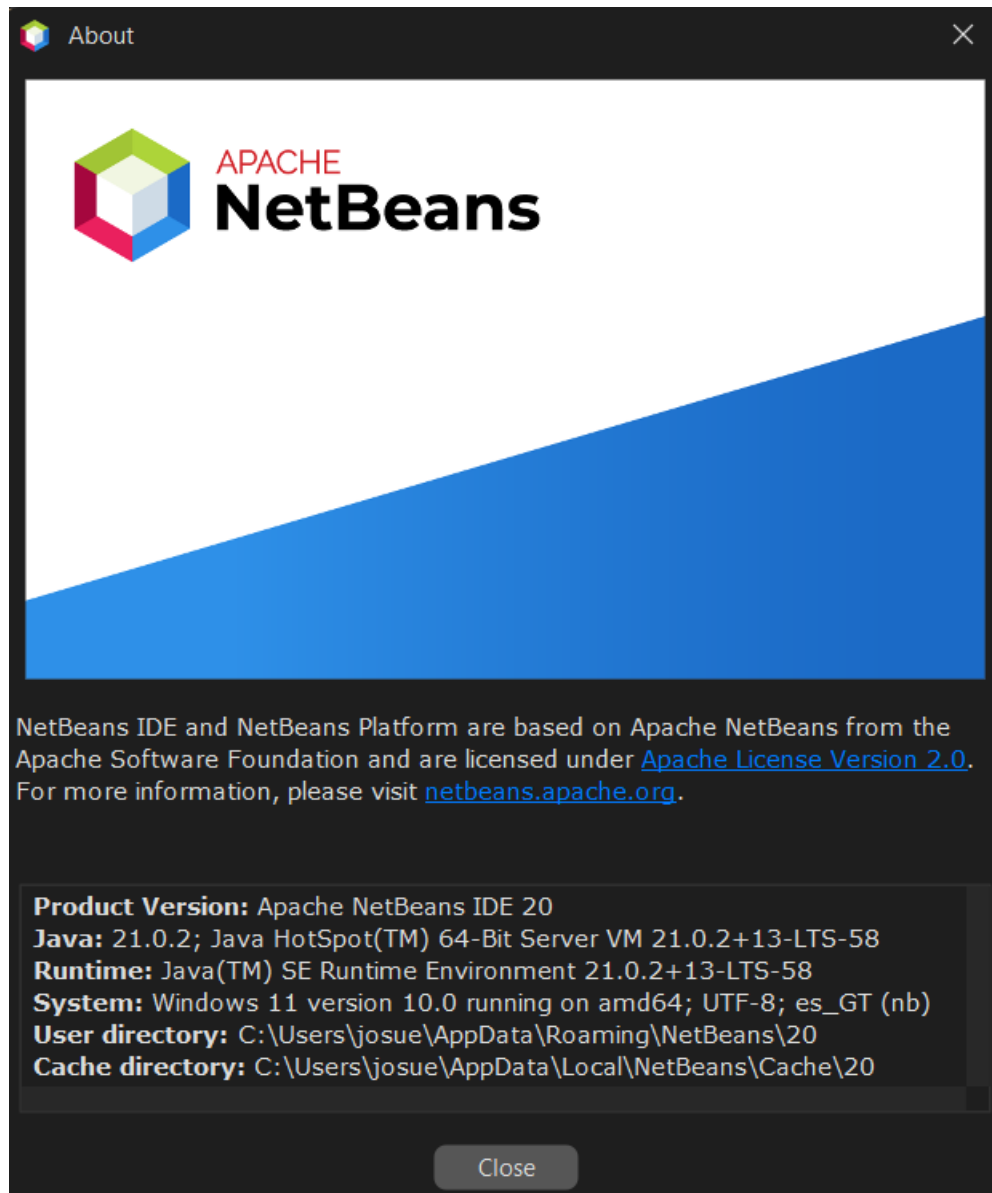
Fecha: 10/03/2024

Sección: N

Auxiliar: Walter Alexander Guerra Duque

Requerimientos del Sistema




En la realización del software el IDE utilizado fue APACHE NetBeans, con las especificaciones que se observan en la siguiente imagen.


















También se utilizó la versión del jdk de java como se observa en la siguiente imagen.

```
java 21.0.2 2024-01-16 LTS
Java(TM) SE Runtime Environment (build 21.0.2+13-LTS-58)
Java HotSpot(TM) 64-Bit Server VM (build 21.0.2+13-LTS-58, mixed mode, sharing)
```

Para la realización del analizador léxico y sintáctico se utilizaron las librerías jflex, y cup, las versiones las puede visualizar en la siguiente imagen.

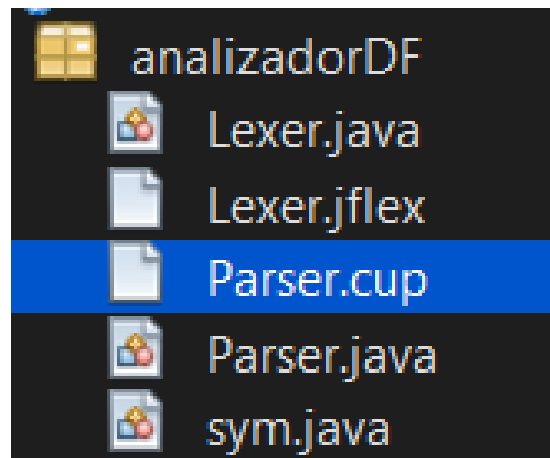
```
>  java-cup-11b-runtime.jar
>  java-cup-11b.jar
>  jflex-1.9.1.jar
```

Para la realización de las gráficas estadísticas, y los cálculos de funciones estadísticas se utilizaron las librerías siguientes.

```
>  commons-math3-3.6.1-javadoc.jar
>  commons-math3-3.6.1-sources.jar
>  commons-math3-3.6.1-test-sources.jar
>  commons-math3-3.6.1-tests.jar
>  commons-math3-3.6.1-tools.jar
>  commons-math3-3.6.1.jar
>  hamcrest-core-1.3.jar
>  jcommon-1.0.23.jar
>  jfreechart-1.0.19-experimental.jar
>  jfreechart-1.0.19-swt.jar
>  jfreechart-1.0.19.jar
>  jfreechart-2.0.jar
>  junit-4.11.jar
>  orsoncharts-1.4-eval-nofx.jar
>  orsonpdf-1.6-eval.jar
>  servlet.jar
>  swtgraphics2d.jar
```

Analizador Léxico y Sintáctico

Para conseguir la solución del programa se realizó analizador léxico, para el reconocimiento de archivos de tipo .df, por lo que se utiliza las librerías mencionadas anteriormente. En la siguiente imagen podrá observar el paquete para el analizador, con sus respectivos archivos.



Dentro del archivo Lexer.jflex se puede observar los caracteres y expresiones regulares y palabras reservadas que serán reconocidos por el lenguaje, así como el manejo de errores léxicos que puedan ocurrir.

```
// -----> Expresiones Regulares

entero = -?[0-9]+
cadena = [""] [^\n]*[""]
//caracter = [''] [^\n]*['']
decimal = -?[0-9]+\.[0-9]+
comentlinea = (\!)(.+)*
comentmultilinea = \<\![^<!]*[!>]*\!\>
id = [A-Za-z_][A-Za-z0-9_]*
```

```

"="      { funcionalidad.Funcion.addTokensDataF(yytext(), "Signo igual", yyline, yycolumn);
          return new Symbol(sym.IGUAL, yycolumn, yyline, yytext()); }
";"      { funcionalidad.Funcion.addTokensDataF(yytext(), "Dos puntos", yyline, yycolumn);
          return new Symbol(sym.PUNTOYCOMA, yycolumn, yyline, yytext()); }
"("      { funcionalidad.Funcion.addTokensDataF(yytext(), "Parentesis abre", yyline, yycolumn);
          return new Symbol(sym.PARENTESIS_A, yycolumn, yyline, yytext()); }
")"      { funcionalidad.Funcion.addTokensDataF(yytext(), "Parentesis cierra", yyline, yycolumn);
          return new Symbol(sym.PARENTESIS_C, yycolumn, yyline, yytext()); }
">"      { funcionalidad.Funcion.addTokensDataF(yytext(), "Mayor que", yyline, yycolumn);
          return new Symbol(sym.MAYOR, yycolumn, yyline, yytext()); }
"<"      { funcionalidad.Funcion.addTokensDataF(yytext(), "Menor que", yyline, yycolumn);
          return new Symbol(sym.MENOR, yycolumn, yyline, yytext()); }
":"      { funcionalidad.Funcion.addTokensDataF(yytext(), "Dos puntos", yyline, yycolumn);
          return new Symbol(sym.DOSPUNTOS, yycolumn, yyline, yytext()); }
","      { funcionalidad.Funcion.addTokensDataF(yytext(), "Signo coma", yyline, yycolumn);
          return new Symbol(sym.COMA, yycolumn, yyline, yytext()); }
"["      { funcionalidad.Funcion.addTokensDataF(yytext(), "Corchete abre", yyline, yycolumn);
          return new Symbol(sym.CORCHETE_A, yycolumn, yyline, yytext()); }
"]"      { funcionalidad.Funcion.addTokensDataF(yytext(), "Corchete cierra", yyline, yycolumn);
          return new Symbol(sym.CORCHETE_C, yycolumn, yyline, yytext()); }
"_"      { funcionalidad.Funcion.addTokensDataF(yytext(), "Guion Medio", yyline, yycolumn);
          return new Symbol(sym.GUION_MEDIO, yycolumn, yyline, yytext()); }
"@       { funcionalidad.Funcion.addTokensDataF(yytext(), "Arroba", yyline, yycolumn);
          return new Symbol(sym.ARROBA, yycolumn, yyline, yytext()); }

```

```

"program" { funcionalidad.Funcion.addTokensDataF(yytext(), "Palabra Reservada", yyline, yycolumn);
          return new Symbol(sym.R_PROGRAM, yycolumn, yyline, yytext()); }
"end"     { funcionalidad.Funcion.addTokensDataF(yytext(), "Palabra Reservada", yyline, yycolumn);
          return new Symbol(sym.R_END, yycolumn, yyline, yytext()); }
"char"    { funcionalidad.Funcion.addTokensDataF(yytext(), "Cadena", yyline, yycolumn);
          return new Symbol(sym.R_CHAR, yycolumn, yyline, yytext()); }
"var"     { funcionalidad.Funcion.addTokensDataF(yytext(), "Declaracion de variables", yyline, yycolumn);
          return new Symbol(sym.R_VAR, yycolumn, yyline, yytext()); }
"double"  { funcionalidad.Funcion.addTokensDataF(yytext(), "Decimal", yyline, yycolumn);
          return new Symbol(sym.R_DOUBLE, yycolumn, yyline, yytext()); }
"arr"     { funcionalidad.Funcion.addTokensDataF(yytext(), "Id Arreglos", yyline, yycolumn);
          return new Symbol(sym.R_ARR, yycolumn, yyline, yytext()); }
"sum"     { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada Suma", yyline, yycolumn);
          return new Symbol(sym.R_SUM, yycolumn, yyline, yytext()); }
"res"     { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada Resta", yyline, yycolumn);
          return new Symbol(sym.R_RES, yycolumn, yyline, yytext()); }
"mul"     { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada Multiplicacion", yyline, yycolumn);
          return new Symbol(sym.R_MUL, yycolumn, yyline, yytext()); }
"div"     { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada Division", yyline, yycolumn);
          return new Symbol(sym.R_DIV, yycolumn, yyline, yytext()); }
"mod"     { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada Modulo", yyline, yycolumn);
          return new Symbol(sym.R_MOD, yycolumn, yyline, yytext()); }
"media"   { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada Media", yyline, yycolumn);
          return new Symbol(sym.R_MEDIA, yycolumn, yyline, yytext()); }
"mediana" { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada Mediana", yyline, yycolumn);
          return new Symbol(sym.R_MEDIANA, yycolumn, yyline, yytext()); }
"moda"    { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada para Moda", yyline, yycolumn);
          return new Symbol(sym.R_MODA, yycolumn, yyline, yytext()); }
"varianza" { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada Varianza", yyline, yycolumn);
          return new Symbol(sym.R_VARIANZA, yycolumn, yyline, yytext()); }
"max"     { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada Maximo", yyline, yycolumn);
          return new Symbol(sym.R_MAX, yycolumn, yyline, yytext()); }
"min"     { funcionalidad.Funcion.addTokensDataF(yytext(), "Reservada Minimo", yyline, yycolumn);
          return new Symbol(sym.R_MIN, yycolumn, yyline, yytext()); }

```

```

(entero)  { funcionalidad.Funcion.addTokensDataF(yytext(), "Double", yyline, yycolumn);
           return new Symbol(sym.ENTERO, yycolumn, yyline, yytext()); }
(cadena)  { funcionalidad.Funcion.addTokensDataF(yytext(), "String", yyline, yycolumn);
           return new Symbol(sym.CADENA, yycolumn, yyline, yytext()); }
(decimal) { funcionalidad.Funcion.addTokensDataF(yytext(), "Double", yyline, yycolumn);
           return new Symbol(sym.DECIMALES, yycolumn, yyline, yytext()); }
(id)      { funcionalidad.Funcion.addTokensDataF(yytext(), "Identificador", yyline, yycolumn);
           return new Symbol(sym.ID, yycolumn, yyline, yytext()); }
(comentlinea) {}
(comentmultilinea) {}

//-----> Ingridados
[ \t\r\n\f]      (/* Espacios en blanco se ignoran */)

//-----> Errores Léxicos
.
    { System.out.println("Error Lexico: " + yytext() + " | Fila:" + yyline + " | Columna: " + yycolumn);
      funcionalidad.Funcion.addErroresLista( "Léxico" , "El carácter \" + yytext() + "\" no pertenece al lenguaje", yyline, yycolumn

```

Dentro del archivo Parser.cup se tiene el manejo de errores sintácticos, la gramática que servirá para verificar la correcta lectura del archivo y la declaración de los terminales y no terminales, también se realiza el manejo de la lógica para almacenar lo reconocido, como declaraciones, impresiones, operaciones, etc.

```

parser code
{
    public String resultado = "";

    public void syntax_error(Symbol s)
    {
        System.err.println("Error Sintactico: " + s.value + " - Fila: " + s.right + " - Columna: " + s.left + ". Recuperado" );
        funcionalidad.Funcion.addErroresLista("Sintactico", (String) s.value, s.right, s.left );
    }

    public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception
    {
        System.err.println("Error Sintactico: " + s.value + " - Fila: " + s.right + " - Columna: " + s.left + ". Sin recuperacion." );
        funcionalidad.Funcion.addErroresLista( "Sintactico", (String) s.value, s.right, s.left );
        JOptionPane.showMessageDialog(null, "Error sintactico: No se pudo recuperar. Por favor corregir el error", (String) s.value, JOptionPane
    }
}

:)

```

```

//-----> Declaración de terminales
terminal String IGUAL;
terminal String PUNTOYCOMA;
terminal String PARENTESIS_A;
terminal String PARENTESIS_C;
terminal String MAYOR;
terminal String MENOR;
terminal String DOSPUNTOS;
terminal String COMA;
terminal String CORCHETE_A;
terminal String CORCHETE_C;
terminal String GUION_MEDIO;
terminal String ARROBA;

```

```
//-----> Declaración de no terminales
non terminal inicio;
non terminal codigo;
non terminal instruccion;
non terminal declaracion;
non terminal numero;
non terminal cadena;
non terminal arreglos;
non terminal valoresnumeros;
non terminal valorescadenas;
non terminal listanum;
non terminal operaciones;
non terminal funcionesop;
non terminal impresiones;
non terminal imprimirexpresiones;
non terminal imprimirarreglos;
non terminal listaexpresiones;
non terminal expresiones;
non terminal funcionesest;
non terminal arreglodouble;
non terminal funcgrafic;
```

```
// -----> Producciones <-----
inicio ::= R_PROGRAM codigo R_END R_PROGRAM      {(): funcionalidad.Funcion.obtenerImpresion();}
;
codigo ::= codigo instruccion
        | instruccion
;
instruccion ::= funcgrafic
              | declaracion
              | arreglos
              | operaciones
              | impresiones
              | error PUNTOYCOMA
;
declaracion ::= R_VAR:pos1 DOSPUNTOS R_DOUBLE DOSPUNTOS DOSPUNTOS ID:nomvariable MENOR GUION_MEDIO numero:valorrecibido R_END PUNTOYCOMA
              | R_VAR:pos2 DOSPUNTOS R_CHAR CORCHETE_A CORCHETE_C DOSPUNTOS DOSPUNTOS ID:nomvariable1 MENOR GUION_MEDIO cadena:datorecibido R_E
;
numero ::= DECIMALES:v1                                {(): RESULT = Double.parseDouble(v1); :}
        | ENTERO:v2                                    {(): RESULT = Integer.valueOf(v2); :}
        | ID:v3                                         {(): RESULT = Double.parseDouble(funcionalidad.Funcion.buscarValo
;
cadena ::= CADENA:v4                                    {(): RESULT = v4.replace("\",", ""); :}
        | ID:v5                                         {(): RESULT = funcionalidad.Funcion.buscarValordId("decVariables"
;
arreglos ::= R_ARR:pos3 DOSPUNTOS R_DOUBLE DOSPUNTOS DOSPUNTOS ARROBA ID:ncmbrevariable2 MENOR GUION_MEDIO CORCHETE_A valoresnumeros:valorrecib
        | R_ARR:pos4 DOSPUNTOS R_CHAR CORCHETE_A CORCHETE_C DOSPUNTOS DOSPUNTOS ARROBA ID:ncmbrevariable3 MENOR GUION_MEDIO CORCHETE_A valoresc
        | R_ARR:pos5 DOSPUNTOS R_DOUBLE DOSPUNTOS DOSPUNTOS ARROBA ID:ncmbrevariable4 MENOR GUION_MEDIO ARROBA ID:v6 R_END PUNTOYCOMA
        | R_ARR:pos6 DOSPUNTOS R_CHAR CORCHETE_A CORCHETE_C DOSPUNTOS DOSPUNTOS ARROBA ID:ncmbrevariable5 MENOR GUION_MEDIO ARROBA ID:v7 R_END
```

Manejo de Errores

Para el manejo de los errores, se utilizó un objeto que contendría los parámetros que se observan en la imagen y estas a su vez se almacenaron dentro de una lista y así poder recorrerla para los reportes.

```

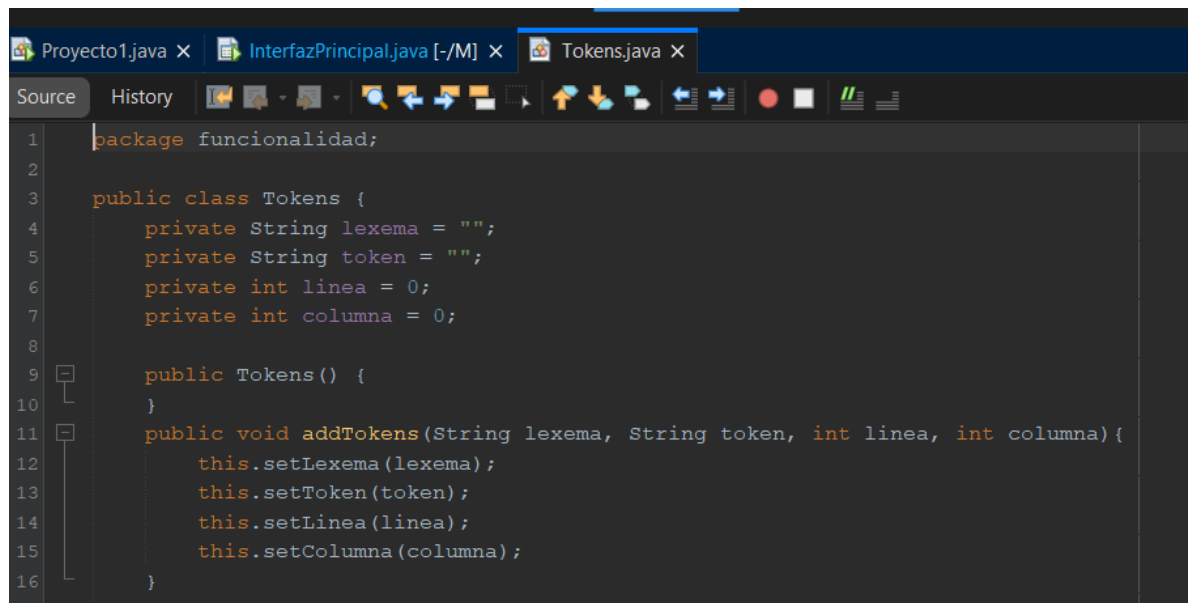
public class Errores {
    private String tipo = "";
    private String descripcion = "";
    private int linea = 0;
    private int columna = 0;

    public Errores() {
    }
    public void addErrores(String tipo, String descripcion, int linea, int columna){
        this.setTipo(tipo);
        this.setDescripcion(descripcion);
        this.setLinea(linea);
        this.setColumna(columna);
    }
}

```

Manejo de Tokens

También se manejaron a través de objetos, y se almacenaron en una lista para poder utilizarlo en el reporte correspondiente, el objeto cuenta con los siguientes parámetros:



```

package funcionalidad;

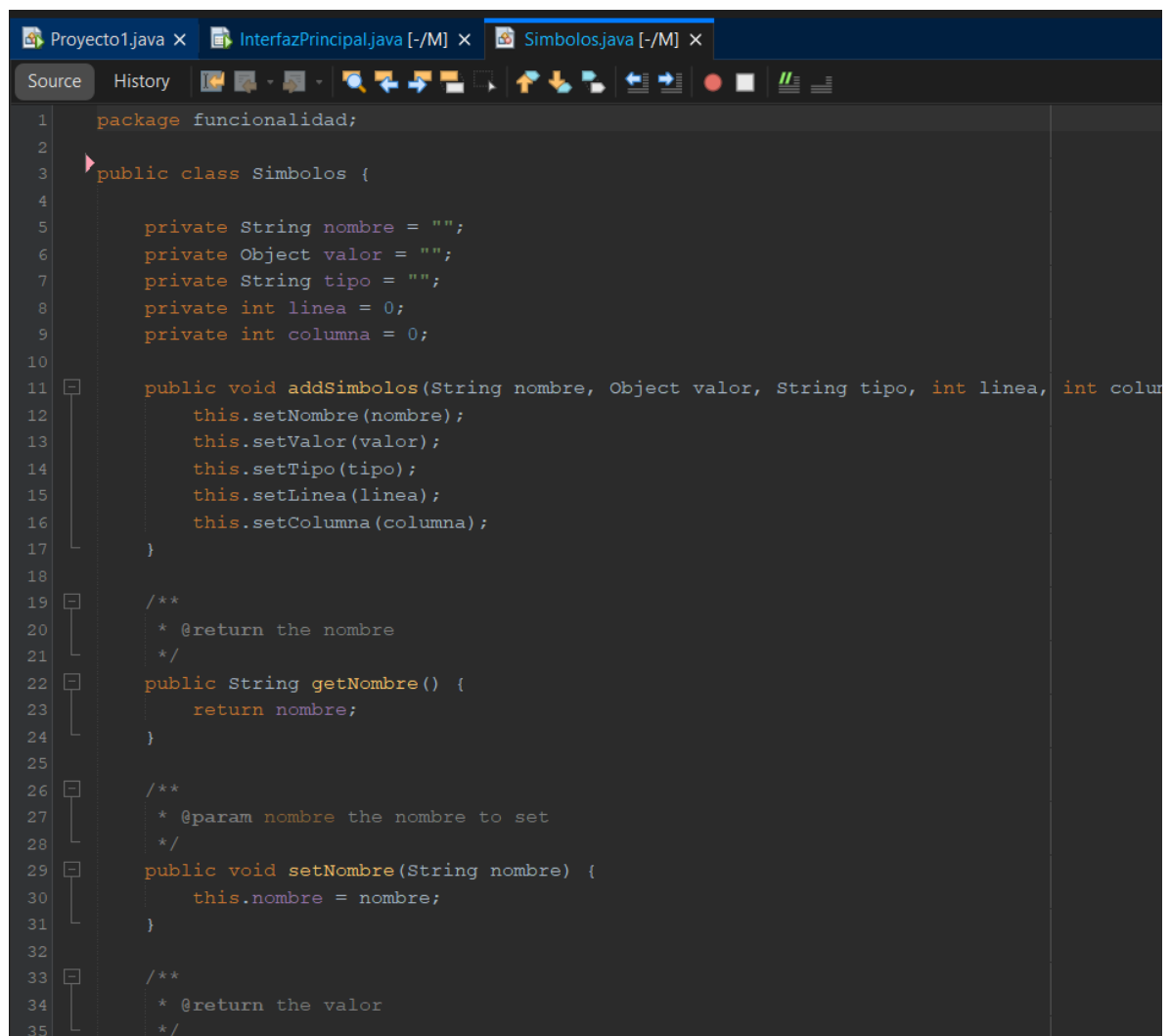
public class Tokens {
    private String lexema = "";
    private String token = "";
    private int linea = 0;
    private int columna = 0;

    public Tokens() {
    }
    public void addTokens(String lexema, String token, int linea, int columna){
        this.setLexema(lexema);
        this.setToken(token);
        this.setLinea(linea);
        this.setColumna(columna);
    }
}

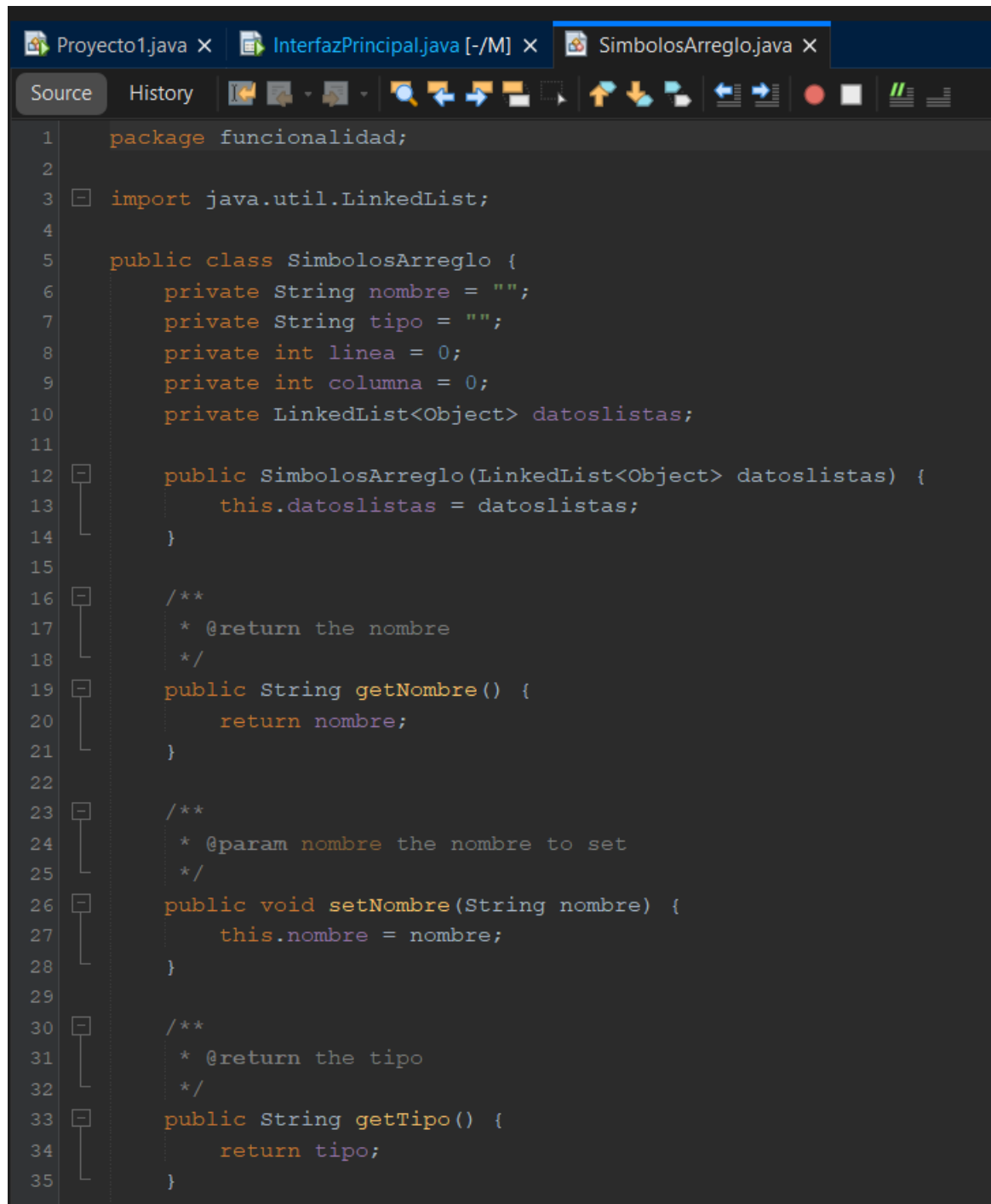
```


Manejo de Simbolos

Para este se manejan dos objetos, el primero para almacenar datos con valores simples, y el segundo para manejar los que incluyen arreglos de valores, y todos estos objetos se almacenaron a su vez en hashmap y linkedlist para posteriormente poder ser utilizado para obtener los datos tanto para manejo de funcionalidades y el reporte. Para ello se muestran los dos objetos en las imágenes siguientes:



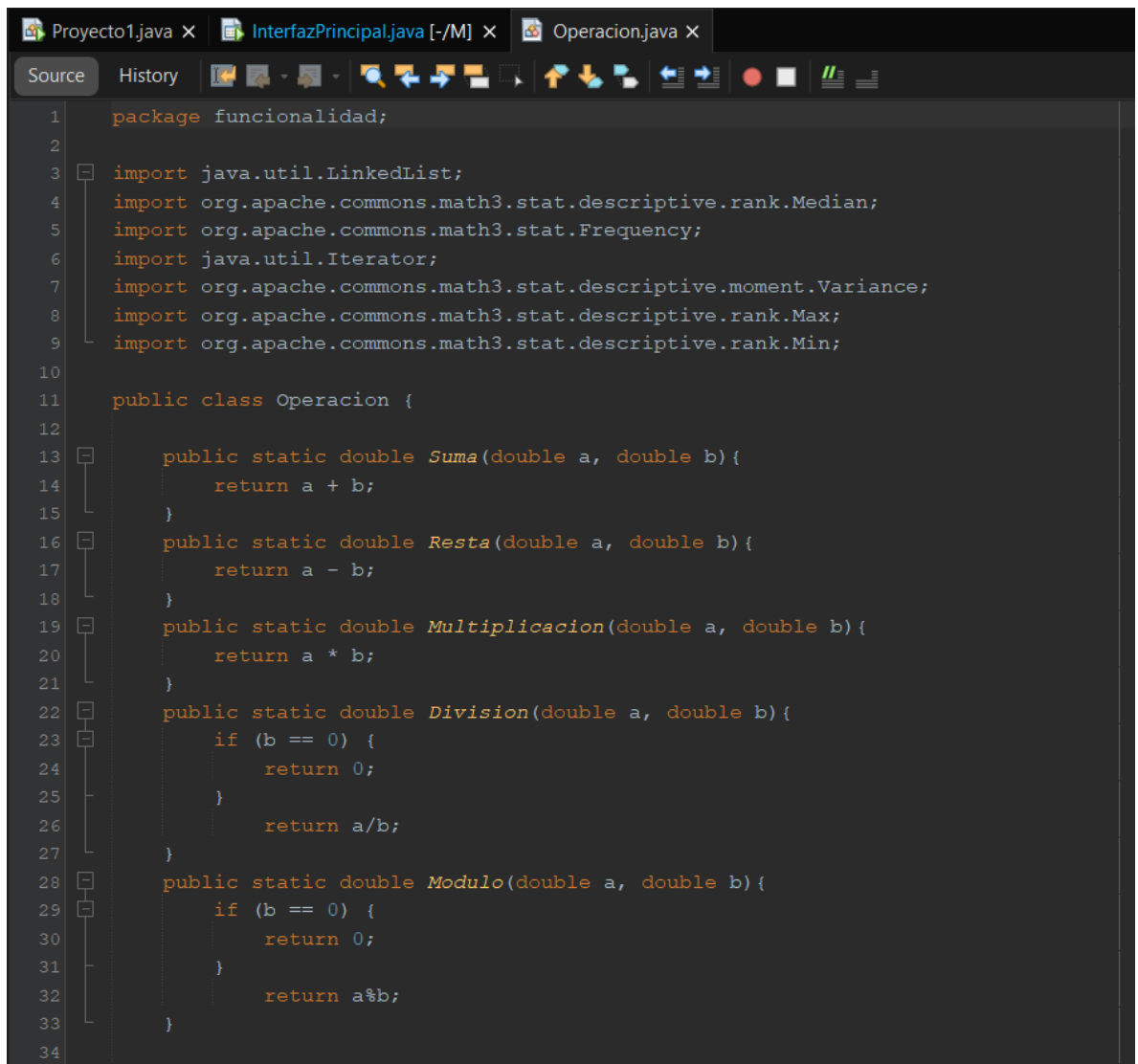
```
1 package funcionalidad;
2
3 public class Simbolos {
4
5     private String nombre = "";
6     private Object valor = "";
7     private String tipo = "";
8     private int linea = 0;
9     private int columna = 0;
10
11     public void addSimbolos(String nombre, Object valor, String tipo, int linea, int columna) {
12         this.setNombre(nombre);
13         this.setValor(valor);
14         this.setTipo(tipo);
15         this.setLinea(linea);
16         this.setColumna(columna);
17     }
18
19     /**
20      * @return the nombre
21      */
22     public String getNombre() {
23         return nombre;
24     }
25
26     /**
27      * @param nombre the nombre to set
28      */
29     public void setNombre(String nombre) {
30         this.nombre = nombre;
31     }
32
33     /**
34      * @return the valor
35      */
```



```
1 package funcionalidad;
2
3 import java.util.LinkedList;
4
5 public class SimbolosArreglo {
6     private String nombre = "";
7     private String tipo = "";
8     private int linea = 0;
9     private int columna = 0;
10    private LinkedList<Object> datoslistas;
11
12    public SimbolosArreglo(LinkedList<Object> datoslistas) {
13        this.datoslistas = datoslistas;
14    }
15
16    /**
17     * @return the nombre
18     */
19    public String getNombre() {
20        return nombre;
21    }
22
23    /**
24     * @param nombre the nombre to set
25     */
26    public void setNombre(String nombre) {
27        this.nombre = nombre;
28    }
29
30    /**
31     * @return the tipo
32     */
33    public String getTipo() {
34        return tipo;
35    }
36}
```

Manejo de Operaciones

Para las operaciones aritméticas y estadísticas se realizó una clase donde se tuviesen las funciones que reciben como parámetros los datos y los operan internamente, para así retornar los resultados, incluso se utilizó la librería commons.math3, de apache para las operaciones de funciones estadísticas como moda, mediana, varianza, etc.



```
1 package funcionalidad;
2
3 import java.util.LinkedList;
4 import org.apache.commons.math3.stat.descriptive.rank.Median;
5 import org.apache.commons.math3.stat.Frequency;
6 import java.util.Iterator;
7 import org.apache.commons.math3.stat.descriptive.moment.Variance;
8 import org.apache.commons.math3.stat.descriptive.rank.Max;
9 import org.apache.commons.math3.stat.descriptive.rank.Min;
10
11 public class Operacion {
12
13     public static double Suma(double a, double b) {
14         return a + b;
15     }
16
17     public static double Resta(double a, double b) {
18         return a - b;
19     }
20
21     public static double Multiplicacion(double a, double b) {
22         return a * b;
23     }
24
25     public static double Division(double a, double b) {
26         if (b == 0) {
27             return 0;
28         }
29         return a/b;
30     }
31
32     public static double Modulo(double a, double b) {
33         if (b == 0) {
34             return 0;
35         }
36         return a%b;
37     }
38 }
```

```
35 public static double Media(LinkedList<Object> recListaD){
36     double cantiV = recListaD.size();
37
38     double suma = 0;
39     for (int i=0; i< recListaD.size();i++){
40         suma += Double.parseDouble(recListaD.get(i).toString());
41     }
42     double resultado = suma/cantiV;
43     return resultado;
44 }
45
46 public static double Mediana(LinkedList<Object> recListaD){
47     double [] temp = new double[recListaD.size()];
48     for (int i = 0; i < recListaD.size(); i++){
49         temp[i]= Double.parseDouble(recListaD.get(i).toString());
50     }
51     Median mediana = new Median();
52     double resultado = mediana.evaluate(temp);
53     return resultado;
54 }
```

```

55 public static double Moda(LinkedList<Object> recListaD){
56     double [] temp = new double[recListaD.size()];
57     for (int i = 0; i < recListaD.size(); i++){
58         temp[i]= Double.parseDouble(recListaD.get(i).toString());
59     }
60     Frequency frequency = new Frequency();
61     for (double d : temp) {
62         frequency.addValue(d);
63     }
64
65     // Encuentra el valor más frecuente (moda)
66     double resultado = 0;
67     long maxFrequency = 0;
68     Iterator<Comparable<?>> iterator = frequency.valuesIterator();
69     while (iterator.hasNext()) {
70         Comparable<?> value = iterator.next();
71         long currentFrequency = frequency.getCount(value);
72         if (currentFrequency > maxFrequency) {
73             resultado = (Double) value;
74             maxFrequency = currentFrequency;
75         }
76     }
77     return resultado;
78 }
79
80 public static double Varianza(LinkedList<Object> recListaD){
81     double [] temp = new double[recListaD.size()];
82     for (int i = 0; i < recListaD.size(); i++){
83         temp[i]= Double.parseDouble(recListaD.get(i).toString());
84     }
85     Variance variance = new Variance();
86     double resultado = variance.evaluate(temp);
87     return resultado;
88 }

```

```

89     public static double Max(LinkedList<Object> recListaD){
90         double [] temp = new double[recListaD.size()];
91         for (int i = 0; i < recListaD.size(); i++){
92             temp[i]= Double.parseDouble(recListaD.get(i).toString());
93         }
94         Max max = new Max();
95         double resultado = max.evaluate(temp);
96         return resultado;
97     }
98     public static double Min(LinkedList<Object> recListaD){
99         double [] temp = new double[recListaD.size()];
100        for (int i = 0; i < recListaD.size(); i++){
101            temp[i]= Double.parseDouble(recListaD.get(i).toString());
102        }
103        Min min = new Min();
104        double resultado = min.evaluate(temp);
105        return resultado;
106    }
107
108 }
109

```

Manejo de Gráficas

Para tener un manejo adecuado de las gráficas, se realizó una clase para las mismas, en este se manejaron variables de listas y strings, para almacenar los valores y nombres de títulos, etc que servirían para las mismas.

```

24     public class Grafica {
25
26         //ALMACENANDO NOMBRES DE IMAGENES
27         public static LinkedList<String> listImagenes = new LinkedList<String>();
28         //PARA GRAFICA DE BARRAS
29         public static String tituloBarras;
30         public static String tituloxBarras;
31         public static String tituloYBarras;
32         static LinkedList<Object> listEjexBarras = new LinkedList<Object>();
33         static LinkedList<Object> listEjeyBarras = new LinkedList<Object>();
34
35         //PARA GRAFICA DE PIE
36         public static String tituloPie;
37         static LinkedList<Object> listValuesPie = new LinkedList<Object>();
38         static LinkedList<Object> listLabelPie = new LinkedList<Object>();
39
40         //PARA GRAFICA DE LINE
41         public static String tituloLine;
42         public static String tituloxLine;
43         public static String tituloYLine;
44         static LinkedList<Object> listEjexLine = new LinkedList<Object>();
45         static LinkedList<Object> listEjeyLine = new LinkedList<Object>();
46
47         //PARA HISTOGRAMA
48         public static String tituloHisto;
49         static LinkedList<Object> listValuesHisto = new LinkedList<Object>();

```

Las gráficas acceden a los datos a través de un método en el cual se agregan los valores y títulos necesarios, y al crearse la grafica se almacena como imagen para así poder ser visualizado en el programa.

```
59 public static void barras() {
60     try {
61         //Ingreso de datos
62         DefaultCategoryDataset dataset = new DefaultCategoryDataset();
63
64         LinkedList<Object> listasimbolos1 = new LinkedList<Object>();
65         listasimbolos1 = listEjexBarras;
66         LinkedList<Object> listasimbolos2 = new LinkedList<Object>();
67         listasimbolos2 = listEjeyBarras;
68         for (int i = 0; i < listasimbolos2.size(); i++) {
69             String d = (String) listasimbolos1.get(i);
70             Double c = Double.parseDouble(listasimbolos2.get(i).toString());
71             dataset.addValue(c, "Valor", d);
72         }
73
74         // Creación de gráfica
75         JFreeChart grafica
76             = ChartFactory.createBarChart(
77                 tituloBarras, //TITULO
78                 tituloxBarras, tituloyBarras,
79                 dataset,
80                 PlotOrientation.VERTICAL,
81                 true, true, true);
82
83         // Mostrar
84         // ChartFrame frame = new ChartFrame("Grafica de Barras", grafica);
85         // frame.pack();
86         // frame.setVisible(true);
87         // Guardar la gráfica como una imagen
88         int width = 580;
89         /* Width of the image */
90         int height = 470;
91         /* Height of the image */
92         File archivoImagen = new File("graficaBARRA.png");
93         listImagenes.add("graficaBARRA.png");
```

```

108 public static void gPie() {
109     try {
110         //Ingreso de datos
111         DefaultPieDataset dataset = new DefaultPieDataset();
112
113         LinkedList<Object> listasimbolos1 = new LinkedList<Object>();
114         listasimbolos1 = listLabelPie;
115         LinkedList<Object> listasimbolos2 = new LinkedList<Object>();
116         listasimbolos2 = listValuesPie;
117         for (int i = 0; i < listasimbolos2.size(); i++) {
118             String d = (String) listasimbolos1.get(i);
119             Double c = Double.valueOf(listasimbolos2.get(i).toString());
120             dataset.setValue(d, c);
121         }
122         /*for(int i = 0; i < 5; i++){
123             dataset.setValue(ejex[i], valores[i]);
124         }*/
125         // Creación de gráfica
126         JFreeChart grafica = ChartFactory.createPieChart(tituloPie, dataset);
127         PiePlot plot = (PiePlot) grafica.getPlot();
128         plot.setLabelGenerator(new StandardPieSectionLabelGenerator("{0}: {1}"));
129         plot.setLabelBackgroundPaint(new Color(220, 220, 220)); // Configura el color de fondo de las etiquetas
130         plot.setLabelOutlinePaint(Color.BLACK); // Configura el color del borde de las etiquetas
131         plot.setLabelShadowPaint(Color.WHITE); // Configura el color de la sombra de las etiquetas
132
133         // Mostrar
134         int width = 580;
135         /* Width of the image */
136         int height = 470;
137         /* Height of the image */
138         File archivoImagen = new File("graficaPIE.png");
139         listImagenes.add("graficaPIE.png");
140         ChartUtilities.saveChartAsPNG(archivoImagen, grafica, width, height);
141     } catch (IOException ex) {

```

```

154 public static void gLine() {
155     try {
156         // Ingreso de datos
157         DefaultCategoryDataset dataset = new DefaultCategoryDataset();
158
159         // Agregar datos al dataset
160         for (int i = 0; i < listEjexLine.size(); i++) {
161             dataset.addValue(Double.parseDouble(listEjeyLine.get(i).toString()), "Datos", listEjexLine.get(i).toString());
162         }
163
164         // Creación de la gráfica de líneas
165         JFreeChart chart = ChartFactory.createLineChart(tituloLine, tituloxLine, tituloyLine, dataset, PlotOrientation.VERTICAL, true, true);
166
167         // Mostrar
168         int width = 580;
169         /* Width of the image */
170         int height = 470;
171         File chartFile = new File("graficaLinea.png");
172         listImagenes.add("graficaLinea.png");
173         ChartUtilities.saveChartAsPNG(chartFile, chart, width, height);
174     } catch (IOException ex) {
175         Logger.getLogger(Grafica.class.getName()).log(Level.SEVERE, null, ex);
176     }
177 }

```

Para el histograma fue necesario realizar otras operaciones, las cuales son la frecuencia, frecuencia acumulada y frecuencia relativa, estas a su vez se almacenan en una variable que servirá para realizar la impresión en consola y luego se crea la imagen de la gráfica.


```

10 public class Funcion {
11
12     static LinkedList<Object> listaTokensDataF = new LinkedList<Object>();
13     //Para guardar los errores
14     static LinkedList<Object> listaErrores = new LinkedList<Object>();
15
16     //Para almacenar los datos en simbolos
17     static Map<String, LinkedList<Object>> hashMapSimbolos = new HashMap<>();
18     static String nombreArchivo = "";
19     public static String txtSalida = "";
20     /*
21     Trabajando con el codigo siguiente
22     */
23
24     //Para almacenar los datos para las impresiones
25     static LinkedList<String> listImprimir = new LinkedList<String>();

```

```

27 //METODO PARA OBTENER LA LISTA DE IMPRESION
28 public static void obtenerLImpresion() {
29     if (listImprimir != null) {
30         for (int i = 0; i < listImprimir.size(); i++) {
31             txtSalida += listImprimir.get(i);
32             txtSalida += "\n";
33         }
34     } else {
35         txtSalida = "";
36     }
37 }
38
39 //METODO PARA ALMACENAR LAS IMPRESIONES
40 public static void addImpresiones(LinkedList<Object> lmpresiones) {
41     String texto = "!Salida: ";
42     for (int i = 0; i < lmpresiones.size(); i++) {
43         texto += lmpresiones.get(i).toString() + " , ";
44     }
45     if (texto.lastIndexOf(",") != -1) {
46         // Encontrar la posición de la última coma
47         int indiceUltimaComa = texto.lastIndexOf(",");
48
49         // Crear una subcadena que excluya la última coma
50         texto = texto.substring(0, indiceUltimaComa) + texto.substring(indiceUltimaComa + 1);
51     }
52     // System.out.println(texto);
53     listImprimir.add(texto);
54 }

```

```

56 public static void addImpresionesCol(String stitulo, LinkedList<Object> limpresiones) {
57     String texto = "-----\n\t";
58     texto += stitulo + "\n-----\n\t";
59     for (int i = 0; i < limpresiones.size(); i++) {
60         texto += limpresiones.get(i).toString() + "\n";
61     }
62     listImprimir.add(texto);
63 }
64
65 public static void addImpresionSimpleH(String txtHisto) {
66     listImprimir.add(txtHisto);
67 }
68
69 //METODO PARA AGREGAR UNA DECLARACIÓN A LA TABLA DE SIMBOLOS
70 public static void addHMSimbolos(String nombrelista, String nombre, Object valor, String tipo, int linea, int columna) {
71     Simbolos sim = new Simbolos();
72     sim.addSimbolos(nombre.toLowerCase(), valor, tipo, linea, columna);
73     hashMapSimbolos.computeIfAbsent(nombrelista, key -> new LinkedList<>()).add(sim);
74 }
75
76 //METODO PARA AGREGAR UNA DECLARACION DE ARREGLOS A LA TABLA DE SIMBOLOS
77 public static void addHMSimbolosA(String nombrelista, String nombre, LinkedList<Object> recListainterna, String tipo, int linea, int columna) {
78     SimbolosArreglo sim = new SimbolosArreglo(recListainterna);
79     sim.setNombre(nombre.toLowerCase());
80     sim.setTipo(tipo);
81     sim.setLinea(linea);
82     sim.setColumna(columna);
83     hashMapSimbolos.computeIfAbsent(nombrelista, key -> new LinkedList<>()).add(sim);
84 }

```

También funciones que sirven para la búsqueda de los valores de los id.

```

86 // METODO PARA OBTENER EL VALOR DE UNA DECLARACION
87 public static Object buscarValordId(String nombrelista, String nombroid) {
88     LinkedList<Object> listasimbolos1 = new LinkedList<Object>();
89     listasimbolos1 = hashMapSimbolos.get(nombrelista);
90     for (int i = 0; i < listasimbolos1.size(); i++) {
91         Simbolos simRecorrerr = (Simbolos) listasimbolos1.get(i);
92         if (nombroid.equals(simRecorrerr.getNombre())) {
93             //System.out.println(i + ". " + simRecorrerr.getNombre() + " - " + simRecorrerr.getTipo() + " - " + simRecorrerr.getValor() +
94             return simRecorrerr.getValor();
95         }
96     }
97     return "Valor no encontrado";
98 }
99
100
101 //METODO PARA OBTENER EL VALOR DE UNA DECLARACION DE UN ARREGLO
102 public static LinkedList<Object> buscarValordIdArr(String nombrelista, String nombroid) {
103     LinkedList<Object> listasimbolos1 = new LinkedList<Object>();
104     listasimbolos1 = hashMapSimbolos.get(nombrelista);
105     for (int i = 0; i < listasimbolos1.size(); i++) {
106         SimbolosArreglo simRecorrerr1 = (SimbolosArreglo) listasimbolos1.get(i);
107         if (nombroid.equals(simRecorrerr1.getNombre())) {
108             LinkedList<Object> listaaux = new LinkedList<Object>();
109             listaaux = simRecorrerr1.getDatoslistas();
110             return listaaux;
111         }
112     }
113     return null;
114 }

```

```

146 public static void addTokensDataF(String lexema, String token, int linea, int columna) {
147     Tokens objToken = new Tokens();
148     objToken.addTokens(lexema, token, linea, columna);
149     listaTokensDataF.add(objToken);
150 }

```

Las funciones para realizar los reportes correspondientes a símbolos, tokens y errores.

```

152 public static void crearReporteTokensDataF() throws FileNotFoundException {
153     FileOutputStream rep = new FileOutputStream("ReporteTokens.html");
154     PrintStream t = new PrintStream(rep);
155     t.println("""
156         <!DOCTYPE html>
157         <html>
158         <style>
159         body{
160             background: #ADA996; /* fallback for old browsers */
161             background: -webkit-linear-gradient(to right, #EAEAEA, #DBDBDB, #F2F2F2, #ADA996); /* Chrome 10-25, Safari 5.1-6 */
162             background: linear-gradient(to right, #EAEAEA, #DBDBDB, #F2F2F2, #ADA996); /* W3C, IE 10+/ Edge, Firefox 16+, Chrome 26+, Op
163         }
164         table, th, td {
165             border:1px solid black;
166         }
167         th{
168             background-color: #a2ab58;
169         }
170         </style>
171         <body>
172
173         <h2> ***** Reporte Tabla de Tokens *****</h2>
174
175         <table style="width:100%">
176         <tr>
177             <th>#</th>
178             <th>Lexema</th>
179             <th>Tipo</th>
180             <th>Linea</th>
181             <th>Columna</th>
182         </tr>\n
183         """);
184         //txt.concat("");
185         int contador = 1;

```

```

175         <table style="width:100%">
176         <tr>
177             <th>#</th>
178             <th>Lexema</th>
179             <th>Tipo</th>
180             <th>Linea</th>
181             <th>Columna</th>
182         </tr>\n
183         """);
184         //txt.concat("");
185         int contador = 1;
186         for (int i = 0; i < listaTokensDataF.size(); i++) {
187             Tokens simRecorrerr = (Tokens) listaTokensDataF.get(i);
188             t.println("""
189                 <tr>
190                     <td>"" + Integer.toString(contador) + "</td>\n"
191                     + "<td>" + simRecorrerr.getLexema() + "</td>\n"
192                     + "<td>" + simRecorrerr.getToken() + "</td>\n"
193                     + "<td>" + simRecorrerr.getLinea() + "</td>\n"
194                     + "<td>" + simRecorrerr.getColumna() + "</td>\n"
195                     + "</tr>\n");
196             contador = contador + 1;
197         }
198         t.println("""
199             </table>
200
201             </body>
202             </html>""");
203         t.close();
204         listaTokensDataF.clear();
205     }

```

```

213 public static void recorrerListaErrores() throws FileNotFoundException { ...55 lines }
268
269 public static void crearReporteSimbolosDataF() throws FileNotFoundException { ...74 lines }

```

Y el método que limpia todas las variables.

```

339     public static void limpiarDatos() {
340         listaTokensDataF.clear();
341         listaErrores.clear();
342         hashMapSimbolos.clear();
343         txtSalida = "";
344         listImprimir.clear();
345     }
346

```

Interfaz Grafica

Dentro de la interfaz se manejan las funcionalidades como el nuevo archivo, abrir archivo, guardar, guardar como, eliminar pestaña, ejecutar, ver reportes. Dentro del código se utilizó JTextArea para el ingreso de los datos, y la salida de los mismos, junto con panel y scroll para poder visualizar los archivos cuando más contenido tuviese, también se utilizó JTabbedPane el cual sirve para realizar las pestañas, para las gráficas se mostraron por medio de un label el cual funciona para enviar ImageIcon y así mostrar las imágenes y para cambiarlas se utilizaron los botones siguiente y anterior. Para ello se utilizaron los siguientes métodos.

```

private void nuevoArchivoActionPerformed(java.awt.event.ActionEvent evt) {
    mPestanas.getModel().clearSelection();
    int tab = mPestanas.getTabCount(); // Obtener la cantidad de pestañas (esto para agregar el nombre nuevo a la pestaña)

    // Crear un nuevo JTextArea y JScrollPane
    JTextArea textArea = new JTextArea();
    textArea.setLineWrap(true); // Esto hace que las líneas largas se envuelvan automáticamente
    textArea.setBackground(Color.LIGHT_GRAY);
    textArea.setWrapStyleWord(true);
    JScrollPane scrollPane = new JScrollPane(textArea);

    // Agregar JScrollPane con JTextArea dentro al panel
    JPanel pane = new JPanel(new BorderLayout()); // Establecer BorderLayout al panel
    pane.add(scrollPane, BorderLayout.CENTER);

    // Añadir la nueva pestaña al JTabbedPane
    mPestanas.insertTab("Nueva Pestaña", null, pane, "Ronda " + tab, tab); // Insertar una nueva pestaña
    //
    ctn += 1;
    mPestanas.setSelectedIndex(tab); // Seleccionar la nueva pestaña creada
}

```

```

335 private void guardarArchivoActionPerformed(java.awt.event.ActionEvent evt) {
336     // Obtener el índice de la pestaña seleccionada
337     int index = mPestanas.getSelectedIndex();
338
339     // Obtener el componente en la pestaña seleccionada
340     Component selectedComponent = mPestanas.getSelectedComponent();
341
342     // Verificar si el componente seleccionado es un contenedor
343     if (selectedComponent instanceof Container) {
344         Container container = (Container) selectedComponent;
345
346         // Buscar el JTextArea dentro del contenedor
347         JTextArea textArea = findTextArea(container);
348         container.getName();
349         if (textArea != null) {
350             // Obtener el texto del JTextArea
351             String texto = textArea.getText();
352             if (texto.equals("")) {
353                 JOptionPane.showMessageDialog(this, "No hay datos para guardar.");
354             } else {
355                 try {
356                     //
357                     String filePath = filePathsMap.get(index);
358                     String filePath = mPestanas.getTitleAt(index);
359                     if (filePath.equals("Nueva Pestaña")) {
360                         JOptionPane.showMessageDialog(this, "No se encontró ninguna ruta de archivo asociada con la pestaña");
361                     } else {
362                         String filePath1 = "C:\\Users\\josue\\OneDrive\\Documentos\\USAC\\Compi 1\\Lab\\OLC1_Proyecto1_2020\\";
363                         FileWriter escribir = new FileWriter(filePath1);
364                         escribir.write(texto);
365                         escribir.close();
366                     }
367                 } catch (IOException e) {
368                     JOptionPane.showMessageDialog(this, "No se puede guardar el archivo ya que no hay ninguno abierto, ut

```

```

381 private void guardarComoActionPerformed(java.awt.event.ActionEvent evt) {
382     JFileChooser elegirArchivo = new JFileChooser();
383     int valuar = elegirArchivo.showSaveDialog(this);
384     if (valuar == JFileChooser.APPROVE_OPTION) {
385         //Obtiene la ruta del archivo junto con la extensión ingresada
386         File selectFile = elegirArchivo.getSelectedFile();
387         //Escribiendo el archivo en la ruta
388         try {
389             // Obtener el índice de la pestaña seleccionada
390             int index = mPestanas.getSelectedIndex();
391
392             // Obtener el componente en la pestaña seleccionada
393             Component selectedComponent = mPestanas.getSelectedComponent();
394
395             // Verificar si el componente seleccionado es un contenedor
396             if (selectedComponent instanceof Container) {
397                 Container container = (Container) selectedComponent;
398
399                 // Buscar el JTextArea dentro del contenedor
400                 JTextArea textArea = findTextArea(container);
401
402                 if (textArea != null) {
403                     // Obtener el texto del JTextArea
404                     String texto = textArea.getText();
405                     FileWriter escribir = new FileWriter(selectFile);
406                     escribir.write(texto);
407                     escribir.close();
408                 } else {
409                     JOptionPane.showMessageDialog(this, "No se encontró un JTextArea en la pestaña seleccionada.");
410                 }
411             } else {
412                 JOptionPane.showMessageDialog(this, "El componente seleccionado no es un contenedor.");
413             }
414         } catch (IOException e) {

```

```

421 private void abrirArchivoActionPerformed(java.awt.event.ActionEvent evt) {
422     JFileChooser elegirArchivo = new JFileChooser();
423     FileNameExtensionFilter data = new FileNameExtensionFilter("Tipos", "df");
424     elegirArchivo.setFileFilter(data);
425     // elegirArchivo.addChoosableFileFilter(data);
426     int valuado = elegirArchivo.showOpenDialog(this);
427
428     if (valuado == JFileChooser.APPROVE_OPTION) {
429         // System.out.println("Archivo Seleccionado: " + elegirArchivo.getSelectedFile().getName());
430         archivoElegido = elegirArchivo.getSelectedFile().getName();
431         //nombreArchivo.setText(archivoElegido);
432         //*****Espacio para multiples pestañas
433         mPestanas.getModel().clearSelection();
434         // Comprueba si la pestaña seleccionada es la primera pestaña (utilizada para agregar más pestañas)
435         int tab = mPestanas.getTabCount(); // Obtiene la cantidad de pestañas (esto para agregar el nombre nuevo a la pestaña)
436
437         JPanel pane = new JPanel(); // JPanel de prueba
438         // Básicamente aquí debes añadir el contenido de la nueva pestaña que desees
439
440         //*****
441         File selectFile = elegirArchivo.getSelectedFile();
442         // filePathsMap.put(tab, selectFile.getAbsolutePath().toString());
443         try {
444             BufferedReader leer = new BufferedReader(new InputStreamReader(new FileInputStream(selectFile.getAbsolutePath().toString())));
445             String linea;
446             String parrafo = "";
447             while ((linea = leer.readLine()) != null) {
448                 parrafo += linea + "\n";
449             }
450             leer.close();
451             // entradaTextoP.setText(parrafo);
452             JTextArea textArea = new JTextArea(parrafo.toString());
453             textArea.setLineWrap(true); // Esto hace que las líneas largas se envuelvan automáticamente
454             textArea.setWrapStyleWord(true);
455             textArea.setBackground(Color.LIGHT_GRAY);

```

```

471 private void realizaEjecucionActionPerformed(java.awt.event.ActionEvent evt) {
472     // Obtener el índice de la pestaña seleccionada
473     int index = mPestanas.getSelectedIndex();
474
475     // Obtener el componente en la pestaña seleccionada
476     Component selectedComponent = mPestanas.getSelectedComponent();
477
478     // Verificar si el componente seleccionado es un contenedor
479     if (selectedComponent instanceof Container) {
480         Container container = (Container) selectedComponent;
481
482         // Buscar el JTextArea dentro del contenedor
483         JTextArea textArea = findTextArea(container);
484
485         if (textArea != null) {
486             // Obtener el texto del JTextArea
487             String texto = textArea.getText();
488             proyecto.pkg1.Proyecto1.analizar(texto);
489         } else {
490             System.out.println("No se encontró un JTextArea en la pestaña seleccionada.");
491         }
492     } else {
493         System.out.println("El componente seleccionado no es un contenedor.");
494     }
495
496     //proyecto.pkg1.Proyecto1.analizar(texto);
497     //salidaConsola.setText(texto);
498     try {
499         salidaConsola.setText(funcionalidad.Funcion.txtSalida);
500         funcionalidad.Funcion.recorrerListaErrores();
501         funcionalidad.Funcion.crearReporteTokensDataF();
502         funcionalidad.Funcion.crearReporteSimbolosDataF();
503         visuImagen(visImagen, funcionalidad.Grafica.listImagenes.get(0));
504         //Se limpian los datos de todo el sistema, a excepción de la lista que almacena los nombres de los archivos d
505         funcionalidad.Funcion.limpiarDatos();

```

```

510 private JTextArea findTextArea(Container container) {
511     for (Component component : container.getComponents()) {
512         if (component instanceof JTextArea) {
513             return (JTextArea) component;
514         } else if (component instanceof Container) {
515             JTextArea textArea = findTextArea((Container) component);
516             if (textArea != null) {
517                 return textArea;
518             }
519         }
520     }
521     return null;
522 }
523
524 private void reporteTokenActionPerformed(java.awt.event.ActionEvent evt) {
525     String filepath = new String("ReporteTokens.html");
526     try {
527         File path = new File(filepath);
528         Desktop.getDesktop().open(path);
529     } catch (IOException e) {
530         e.printStackTrace();
531     } catch (IllegalArgumentException e) {
532         JOptionPane.showMessageDialog(null, "No se pudo encontrar el archivo", "Error", JOptionPane.ERROR_MESSAGE);
533         e.printStackTrace();
534     }
535 }
536
537 private void reporteErroresActionPerformed(java.awt.event.ActionEvent evt) {
538     String filepath = new String("ReporteErrores.html");
539     try {
540         File path = new File(filepath);
541         Desktop.getDesktop().open(path);
542     } catch (IOException e) {
543         e.printStackTrace();
544     } catch (IllegalArgumentException e) {

```

```

545 private void eliminarPestañaActionPerformed(java.awt.event.ActionEvent evt) {
546     // Obtener el indice de la pestaña seleccionada
547     int index = mPestanas.getSelectedIndex();
548
549     if (index != -1) { // Si hay alguna pestaña seleccionada
550         int opcion = JOptionPane.showConfirmDialog(this, "¿Deseas guardar los datos?", "Guardar Datos", JOptionPane.YES_NO_CANCEL_OPTION);
551
552         if (opcion == JOptionPane.YES_OPTION) {
553             // El usuario seleccionó "Sí", realiza la acción de guardar
554             // Llama a tu método para guardar los datos aquí
555             guardarComoActionPerformed(evt);
556         } else if (opcion == JOptionPane.NO_OPTION) {
557             mPestanas.remove(index);
558         }
559     }
560 }
561
562 private void anteriorImagenActionPerformed(java.awt.event.ActionEvent evt) {
563     if (cntImagenes > 0 && cntImagenes < funcionalidad.Grafica.listImagenes.size()) {
564         cntImagenes -= 1;
565         visuImagen(visuImagen, funcionalidad.Grafica.listImagenes.get(cntImagenes));
566     }
567 }
568
569 private void siguienteImagenActionPerformed(java.awt.event.ActionEvent evt) {
570     if (cntImagenes >= 0 && cntImagenes < funcionalidad.Grafica.listImagenes.size() - 1) {
571         cntImagenes += 1;
572         visuImagen(visuImagen, funcionalidad.Grafica.listImagenes.get(cntImagenes));
573     }
574 }
575
576 private void visuImagen(JLabel lb, String ruta) {
577     lb.setIcon(null);

```

```

593 private void visuImagen(JLabel lb, String ruta) {
594     lb.setIcon(null);
595     this.image = new ImageIcon(ruta);
596     this.icon = new ImageIcon(
597         this.image.getImage().getScaledInstance(lb.getWidth(), lb.getHeight(), Image.SCALE_DEFAULT));
598     lb.setIcon(this.icon);
599     this.repaint();
600 }

```