

MANUAL TECNICO

ALLAN GÓMEZ

ORGANIZACION DE LENGUAJES Y COMPILEDORES 1



Introducción

El proyecto tiene como objetivo el desarrollo de una solución de software implementando un analizador léxico(jflex) y un analizador sintáctico(cup) por medio de una gramática definida, así como la implementación de una interfaz demostrando los conocimientos de Java

Descripción General

El programa tiene como funcionalidad leer un archivo de texto, que contenga las información necesaria para el funcionamiento del programa, dicho archivo es analizado por medio de los analizadores léxicos y sintácticos y a su vez cuenta con la función de mostrar los tokens y los posibles errores en el archivo de texto ya que estos estarían siendo identificados por el programa; también se podrá visualizar las graficas tanto de Pie, Barras, Lineal he histogramas.

Sistema Operativo

Al ser creado en lenguaje Java, puede ser usado para cualquier tipo de sistema operativo: Linux, Windows, MacOS, etc. Por lo tanto, se puede ejecutar fácilmente un software, sin preocuparte por el soporte del sistema.



Lenguaje utilizado

Java es un lenguaje de programación versátil, multiplataforma y multiparadigma que se destaca por su portabilidad y robustez. Es ampliamente utilizado por empresas de todo el mundo para desarrollar una amplia variedad de aplicaciones, desde aplicaciones web y móviles hasta aplicaciones empresariales de alto rendimiento.

MÉTODOS PRINCIPALES

Para la interfaz grafica se uso el GUI de java, con el drag and drop para poder hacer mucho mas facil la construccion de la misma, dentro de los metodos mas importantes que tiene es el main para ejecutar el programa

Los metodos para agregar datos a un Hashmap tanto global como individual

Clase Main:

En si la clase se llama Proyecto1_compi, pero es quien funciona como main, ya que tiene como función llamar a la clase Ventana para que esta se ejecute. La GUI utiliza elementos gráficos como iconos, menús e imágenes para facilitar el manejo al usuario



```
● ● ●
1 public static void main(String[] args) {
2
3     Ventana ventana = new Ventana();
4     ventana.setVisible(true);
5     ventana.setLocationRelativeTo(null);
6 }
```

Figura 1 Clase Ventana

Fuente: Elaboración Propia

Métodos de la ventana:

Método abrir archivo:

En este método nos sirve para abrir una ventana de nuestra computadora para buscar al dirección del archivo el cual vamos a trabajar, para ello se usar FileInputStream, en el cual se guarda la ruta y los archivos aceptados son definidos previamente, dentro del método se realiza la validación por si en caso existiera un error, por lo cual se hace uso de try catch, y para mas validaciones se usa if el cual verifica que nuestra ruta no este vacia y que a su vez el archivo se inserte en nuestra entorno de entrada de texto para poder visualizar la información. Para saber si el archivo fue o no cargado con éxito se hizo uso de un messagebox que se lanza independientemente de si el archivo fue cargado o no.



```
1 public String AbrirArchivo(File archivo) {
2     String contenido = "";
3     try {
4         entrada = new FileInputStream(archivo);
5         int ascii;
6         while ((ascii = entrada.read()) != -1) {
7             char caracter = (char) ascii;
8             contenido += caracter;
9         }
10    } catch (Exception e) {
11        System.out.println(e);
12    }
13    return contenido;
14}
15
16}
```

Figura 2 Método Abrir

Fuente: Elaboración Propia

Método guardar:

Con esta función se toman los parámetros para guardar el archivo



```
1 public String GuardarArchivo(File archivo, String Contenido) {
2     String mensaje = null;
3     try {
4         salida = new FileOutputStream(archivo);
5         byte[] bytesTxt = Contenido.getBytes();
6         salida.write(bytesTxt);
7         mensaje = "El archivo se guardo con exito :)";
8     } catch (Exception e) {
9         System.out.println(e);
10    }
11    return mensaje;
12}
```

Figura 3 Método Guardar

Fuente: Elaboración Propia



Método analizar:

En este método esta quizá la parte mas importante del programa a la hora de llamar a un análisis, ya que en esta parte le mandamos lo que es nuestro archivo que previamente cargamos a nuestra entrada de texto, a nuestra clase Scanner, que es el analizador léxico y se encargar de hacer todo el proceso de lectura y identificación de tokens así como de posibles errores. Acá ya tocamos lo que es POO y por ello es bueno tener idea de lo que es la programación orientada a objetos:

```
1 private void buttonEjecutarActionPerformed(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_buttonEjecutarActionPerformed
2     Analizador.LexicalAnalysis scan;
3     Analizador.parser parse;
4     try {
5         // realizar el analisis lexico con el scanner
6         scan = new LexicalAnalysis(new BufferedReader(new StringReader(TextEntrada.getText())));
7         // sintactico con el parser
8         parse = new parser(scan);
9         parse.parse();
10        errores.addAll(scan.errores);
11        errores.addAll(parse.getErrores());
12        tokens.addAll(scan.Lexemas);
13        System.out.println("-----\n\n");
14
15
16        String result = "";
17        for (int i = 0; i < parse.salidas.size(); i++) {
18            result += parse.salidas.get(i) + '\n';
19        }
20        ConsolaSalida.setText(result);
21        ConsolaSalida.setEditable(false);
22    } catch (Exception ex) {
23        System.out.println("Error: " + ex.getMessage());
24    }
25 }
26 }
```

Figura 4 Boton Ejecutar

Fuente: Elaboración Propia

Archivo JFLEX:

El archivo jflex tiene como función el análisis léxico del archivo a través del autómata con sus diversos estados, de tal manera que almacena los tokens así como los errores, pero para este archivo el orden no importa, ni el sentido, solo importa que exista una coincidencia con un patrón antes definido.

```

1 //REGLAS SEMANTICAS
2
3 //-----SECCION DE RESERVADAS Y SIMBOLOS-----
4
5 <YYINITIAL> {PROGRAM} {
6     Lexemas.add(new Recorrido("Lexema: "+yytext(), "PROGRAM",yyline+"",yycolumn+""));
7     return new Symbol(sym.PROGRAM, yyline, yycolumn,yytext());
8 }
9
10 <YYINITIAL> {END} {
11     Lexemas.add(new Recorrido("Lexema: "+yytext(), "END",yyline+"",yycolumn+""));
12     return new Symbol(sym.END, yyline, yycolumn,yytext());
13 }
14
15 <YYINITIAL> {VAR} {
16     Lexemas.add(new Recorrido("Lexema: "+yytext(), "VAR",yyline+"",yycolumn+""));
17     return new Symbol(sym.VAR, yyline, yycolumn,yytext());
18 }
19
20 <YYINITIAL> {CONSOLE} {
21     Lexemas.add(new Recorrido("Lexema: "+yytext(), "CONSOLE",yyline+"",yycolumn+""));
22     return new Symbol(sym.CONSOLE, yyline, yycolumn,yytext());
23 }
24
25 <YYINITIAL> {PRINT} {
26     Lexemas.add(new Recorrido("Lexema: "+yytext(), "PRINT",yyline+"",yycolumn+""));
27     return new Symbol(sym.PRINT, yyline, yycolumn,yytext());
28 }
29
30 <YYINITIAL> {CHAR} {
31     Lexemas.add(new Recorrido("Lexema: "+yytext(), "CHAR",yyline+"",yycolumn+""));
32     return new Symbol(sym.CHAR, yyline, yycolumn,yytext());
33 }
34
35 <YYINITIAL> {DOUBLE} {
36     Lexemas.add(new Recorrido("Lexema: "+yytext(), "DOUBLE",yyline+"",yycolumn+""));
37     return new Symbol(sym.DOUBLE, yyline, yycolumn,yytext());
38 }
39
40 <YYINITIAL> {ARREGLO} {
41     Lexemas.add(new Recorrido("Lexema: "+yytext(), "ARR",yyline+"",yycolumn+""));
42     return new Symbol(sym.ARREGLO, yyline, yycolumn,yytext());
43 }
44
45 <YYINITIAL> {SUMA} {
46     Lexemas.add(new Recorrido("Lexema: "+yytext(), "SUM",yyline+"",yycolumn+""));
47     return new Symbol(sym.SUMA, yyline, yycolumn,yytext());
48 }
49
50 <YYINITIAL> {RESTA} {
51     Lexemas.add(new Recorrido("Lexema: "+yytext(), "RES",yyline+"",yycolumn+""));
52     return new Symbol(sym.RESTA, yyline, yycolumn,yytext());
53 }
54
55 <YYINITIAL> {MULTIPLICACION} {
56     Lexemas.add(new Recorrido("Lexema: "+yytext(), "MUL",yyline+"",yycolumn+""));
57     return new Symbol(sym.MULTIPLICACION, yyline, yycolumn,yytext());
58 }
59
60 <YYINITIAL> {DIVISION} {
61     Lexemas.add(new Recorrido("Lexema: "+yytext(), "DIV",yyline+"",yycolumn+""));
62     return new Symbol(sym.DIVISION, yyline, yycolumn,yytext());
63 }
64
65 <YYINITIAL> {MODULO} {
66     Lexemas.add(new Recorrido("Lexema: "+yytext(), "MOD",yyline+"",yycolumn+""));
67     return new Symbol(sym.MODULO, yyline, yycolumn,yytext());
68 }
69
70 <YYINITIAL> {MEDIA} {
71     Lexemas.add(new Recorrido("Lexema: "+yytext(), "MEDIA",yyline+"",yycolumn+""));
72     return new Symbol(sym.MEDIA, yyline, yycolumn,yytext());
73 }
74
75 <YYINITIAL> {MEDIANA} {
76     Lexemas.add(new Recorrido("Lexema: "+yytext(), "MEDIANA",yyline+"",yycolumn+""));
77     return new Symbol(sym.MEDIANA, yyline, yycolumn,yytext());
78 }

```

Figura 5 Scanner.jflex

Fuente: Elaboración Propia

Archivo cup:

Acá se hizo parte de la traducción mediante código java, y la implementacion del sentido de los tokens previamente definidos en el jflex, de manera que pudiesen coincidir con la gramática del archivo de entrada.

```

1 // Terminales (tokens devueltos por el escáner)
2 terminal String PROGRAM;
3 terminal String ID, CHAR, DOUBLE, ARREGLO;
4 terminal String ESTA, MATE, TERCERAC, PESO, NODULO;
5 terminal String MEDIA, MODA, VARIANZA, MAX, MIN, COLUM, GRAPH, PRBRS, PRPIE, PRLINE, EXEC;
6 terminal String TITULO, LABEL, VALUES, EX_PRK, PRV, HISTORAM;
7 terminal String CORABRE, CORIERA, FLECHA, CORAMIE, ARROBA, PARABRE, PARCIERRA, COMA, IGUAL;
8 terminal String DECIMAL, ID, CAREN;
9 /*NO TERMINALES NODOS*/
10 non terminal INICIO, INSTRUCTIONS, INSTRUCTION, SENTENCE, STATEMENT_VARIABLES, TYPE, EXPRESSION, STATEMENT_PRINT, CONCATENATION, ARREGLO_DECLARATION, ARREGLO_TYPE, LIST_VALUES;
11 non terminal PRINT_ARRAY, TYPE_PRINT_ARRAY, TYPE_VAR, FUNCTION_GRAPH, TYPE_GRAPH, EXPRESSION_GRAPH, TYPE_E3, INSTRUCTION_GRAPH, TITULO_PRINCIPAL, TITULO_PRINCIPAL_RULE, TITULO_E3, LABELS, VALORES, EXEC_GRAPH, E3ES, VALORES;
12 /* The grammar rules */
13 start with INICIO;
14 INICIO ::= INSTRUCTIONS;
15 INSTRUCTIONS ::= PROGRAM INSTRUCTION END_PROGRAM;
16 INSTRUCTION ::= PRINT_ARRAY;
17 /*LO DIVIDI EN MUCHAS SENTENCIAS PARA QUE SEA RECURSIVO POR CADA INSTRUCCION PUEDE VENIR UNA SENTENCIA*/
18 SENTENCE ::= INSTRUCTION SENTENCE
19 | SENTENCE;
20 /*SENTENCIA PUEDE DECLARAR VARIABLES, MOSTRAR POR UN PRINT, DECLARAR LOS ARREGLOS Y AUN FALTAN*/
21 SENTENCE_VARIABLES ::= STATEMENT_PRINT
22 | ARREGLO_DECLARATION
23 | PRINT_ARRAY
24 | EXPRESSION_GRAPH
25 | FUNCTION_GRAPH
26 |
27 |
28 |
29 |
30 |
31 STATEMENT_VARIABLES ::= VAR DOSPUNTOS TYPE_VAR DOSPUNTOS DOSPUNTOS ID:identifier FLECHA EXPRESSION:value_ END_PUNTOMA
32 {
33     Object value = null;
34     if (value instanceof String) {
35         value = value.toString(); // Convertir a cadena explicitamente
36     } else {
37         value = value; // Mantener el valor tal como esta
38     }
39     Mapas.setVariable(identifier.toString(), value);
40     System.out.println("Variable declarada: " + identifier + " con valor: " + value);
41 };
42 |
43 |
44 TYPE_VAR ::= TYPE CORABRE CORIERA
45 | TYPE;
46 TYPE ::= CHAR
47 {
48     System.out.println("Tipo de dato char");
49 }
50 | DOUBLE
51 {
52     System.out.println("Tipo de dato double");
53 }
54 ;
55

```

Figura 6 Parser.cup

Fuente: Elaboración Propia

Clase Recorrido:

Esta clase sirve para el manejo de los tokens

```

1 public class Recorrido{
2     //atributos
3     private String lexema;
4     private String token;
5     private String linea;
6     private String columna;
7
8     //constructor
9     public Recorrido(String lexema, String token, String Linea, String colu
10 mna){
11     this.lexema=lexema;
12     this.token=token;
13     this.linea=linea;
14     this.columna=columna;
15 }
16
17     public String getLexema() {
18         return lexema;
19     }
20
21     public String getToken() {
22         return token;
23     }
24
25     public String getLinea() {
26         return linea;
27     }
28
29     public String getColumna() {
30         return columna;
31     }
32
33
34 }
35
36

```

Figura 7 Clase Recorrido

Fuente: Elaboración Propia



Clase Error:

La clase Token tiene como función almacenar los errores encontrados.

```
1 public class Exception_{
2     //atributos
3     private String tipo;
4     private String descripcion;
5     private String linea ;
6     private String columna;
7
8     //constructor
9     public Exception_(String tipo, String descripcion, String linea, String colum
10 na ){
11     this.tipo=tipo;
12     this.descripcion=descripcion;
13     this.linea=linea;
14     this.columna=columna;
15 }
16 //getters
17 public String getTipo() {
18     return tipo;
19 }
20 public String getDescripcion() {
21     return descripcion;
22 }
23 public String getLinea() {
24     return linea;
25 }
26 public String getColumna() {
27     return columna;
28 }
29
30
31 }
```

Figura 8 Clase Exeption_

Fuente: Elaboración Propia

Clase Graficas:

Esta clase es muy importante para lo que respecta a las graficas ya que en esta clase se definen los métodos los cuales reciben los parámetros enviados desde el cup para las graficas tanto, de barra, de pie, lineal he histograma. dentro de la misma se define un array para guardar las diferentes graficas para luego poder ser mostradas.

Dentro de la misma clase esta el método para poder mostrar las graficas, de manera que se crea un nuevo frame en el cual se muestran todas las graficas almacenadas.



```

1  public class Graficas {
2
3      private static ArrayList<JFreeChart> graficas;
4
5      static {
6          graficas = new ArrayList<>();
7      }
8
9      public static void agregarGraficaDeBarras(String titulo, ArrayList<Object> ejeX, ArrayList<Object> ejeY, String tituloX, String tituloY) {
10         DefaultCategoryDataset dataset = new DefaultCategoryDataset();
11         for (int i = 0; i < ejeX.size(); i++) {
12             double valor = Double.parseDouble(ejeY.get(i).toString());
13             dataset.addValue(valor, titulo, ejeX.get(i).toString());
14         }
15         JFreeChart chart = ChartFactory.createBarChart(titulo, tituloX, tituloY, dataset);
16         graficas.add(chart);
17     }
18
19     public static void agregarGraficaDePie(String titulo, ArrayList<Object> values, ArrayList<Object> labels) {
20         DefaultPieDataset dataset = new DefaultPieDataset();
21         for (int i = 0; i < values.size(); i++) {
22             double valor = Double.parseDouble(values.get(i).toString());
23             dataset.setValue(labels.get(i).toString(), valor);
24         }
25         JFreeChart chart = ChartFactory.createPieChart(titulo, dataset);
26         System.out.println("Grafica generada correctamente");
27         graficas.add(chart);
28     }
29
30     public static void agregarGraficaDeLineas(String titulo, ArrayList<Object> ejeX, ArrayList<Object> ejeY, String tituloX, String tituloY) {
31         DefaultCategoryDataset dataset = new DefaultCategoryDataset();
32         for (int i = 0; i < ejeX.size(); i++) {
33             double valor = Double.parseDouble(ejeY.get(i).toString());
34             dataset.addValue(valor, titulo, ejeX.get(i).toString());
35         }
36         JFreeChart chart = ChartFactory.createLineChart(titulo, tituloX, tituloY, dataset);
37         graficas.add(chart);
38     }
39
40     public static void mostrarGraficas() {
41         JFrame frame = new JFrame("Graficas");
42         frame.setLayout(new GridLayout(1, graficas.size()));
43
44         for (JFreeChart chart : graficas) {
45             ChartPanel chartPanel = new ChartPanel(chart);
46             frame.add(chartPanel);
47         }
48
49         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
50         frame.pack();
51         frame.setVisible(true);
52     }
53 }

```

Figura 8 Clase Graficas

Fuente: Elaboración Propia

Clase Mapas:

Bueno en esta clase, casi que recae todo el peso del programa ya que en ella se estan los metodos para guardar una variable y recuperar el valor de una variable definida en el archivo cup, este ejercicio se hace mediante el uso de hashmaps, por lo que se logra recuperar los valores de manera fácil y guardar los valores de manera sencilla también.



```
● ● ●  
1 public class Mapas {  
2     private static HashMap<String, Object> variables = new HashMap<String, Object>();  
3     private static HashMap<String, ArrayList<Object>> arreglos = new HashMap<String, ArrayList<Object>>();  
4  
5     public static void setVariable(String name, Object value) {  
6         variables.put(name, value);  
7     }  
8  
9     public static Object getVariable(String name) {  
10        return variables.get(name);  
11    }  
12  
13    public static void setArreglo(String name, ArrayList<Object> value) {  
14        arreglos.put(name, value);  
15    }  
16  
17    public static ArrayList<Object> getArreglo(String name) {  
18        return arreglos.get(name);  
19    }  
20  
21    public static void mostrarTodosLosArreglos() {  
22        System.out.println("Arreglos y sus IDs asociados:");  
23        for (String key : arreglos.keySet()) {  
24            System.out.println("ID: " + key);  
25            System.out.println("Arreglo: " + arreglos.get(key));  
26        }  
27    }  
28 }
```

Figura 9 Clase Mapas

Fuente: Elaboración Propia

Clase Operaciones Aritméticas:

En esta clase se realiza todo lo que concierne a operaciones matemáticas básicas, ya que en esta clase se realiza la suma, la resta, la multiplicación, la división y el modulo.

Cada método tiene su propia forma de realizar las operaciones, pero cada método siempre necesita de dos parámetros para poder hacer las operaciones, las operaciones pueden ser recursivas por lo que no hay problema en hacer múltiples operaciones dentro de otras.

```

● ● ●
1 public class OperacionesAritmeticas {
2
3     public double suma(Object num1, Object num2) {
4         if (num1 == null || num1.toString().isEmpty() || num2 == null || num2.toString().isEmpty()) {//para este y todos los demás no tiene que ser vacío o nulo
5             return 0.0; // Retorna cero si alguno de los argumentos es nulo o está vacío
6         } else {
7             return Double.parseDouble(num1.toString()) + Double.parseDouble(num2.toString());
8         }
9     }
10
11    public double resta(Object num1, Object num2) {
12        if (num1 == null || num1.toString().isEmpty() || num2 == null || num2.toString().isEmpty()) {
13            return 0.0; // Retorna cero si alguno de los argumentos es nulo o está vacío
14        } else {
15            return Double.parseDouble(num1.toString()) - Double.parseDouble(num2.toString());
16        }
17    }
18
19    public double multiplicacion(Object num1, Object num2) {
20        if (num1 == null || num1.toString().isEmpty() || num2 == null || num2.toString().isEmpty()) {
21            return 0.0; // Retorna cero si alguno de los argumentos es nulo o está vacío
22        } else {
23            return Double.parseDouble(num1.toString()) * Double.parseDouble(num2.toString());
24        }
25    }
26
27    public double division(Object num1, Object num2) {
28        if (num1 == null || num1.toString().isEmpty() || num2 == null || num2.toString().isEmpty()) {
29            return 0.0; // Retorna cero si alguno de los argumentos es nulo o está vacío
30        } else {
31            double n1 = Double.parseDouble(num1.toString());
32            double n2 = Double.parseDouble(num2.toString());
33            if (n2 == 0) {
34                return 0; // Retorna cero si el divisor es cero
35            } else {
36                return n1 / n2;
37            }
38        }
39    }
40
41    public double modulo(Object num1, Object num2) {
42        if (num1 == null || num1.toString().isEmpty() || num2 == null || num2.toString().isEmpty()) {
43            return 0.0; // Retorna cero si alguno de los argumentos es nulo o está vacío
44        } else {
45            return Double.parseDouble(num1.toString()) % Double.parseDouble(num2.toString());
46        }
47    }
48
49
50 }
51

```

Figura 10 Clase OperacionesAritmeticas

Fuente: Elaboración Propia

Clase Operaciones Estadísticas:

En esta clase al igual que en las operaciones aritméticas, necesita de parámetros en sus métodos para poder hacer las operaciones de lo contrario no podrá devolver los datos que se necesitan, pero a diferencia de los métodos de las operaciones aritméticas, que solo recibían dos parámetros para poder realizar sus operaciones, acá en esta clase cada método necesita de un arreglo de datos para poder hacer las operaciones.

```

1  public class OperacionesEstadisticas {
2
3      public double media(ArrayList<Object> valores) {
4          double suma = 0;
5          // Recorre todos los elementos en el ArrayList
6          for (Object valor : valores) {
7              // Convierte cada objeto a String y luego a Double, sumándolos
8              suma += Double.parseDouble(valor.toString());
9          }
10         // Retorna la media dividida por el tamaño del ArrayList
11         return suma / valores.size();
12     }
13
14     public double mediana(ArrayList<Object> valores) {
15         // Crea una lista de Doubles para almacenar los valores convertidos
16         List<Double> numeros = new ArrayList<>();
17         // Convierte los objetos en el ArrayList a Doubles y los añade a la lista
18         for (Object valor : valores) {
19             numeros.add(Double.parseDouble(valor.toString()));
20         }
21         // Ordena la lista de números
22         numeros.sort(null);
23         // Calcula y devuelve la mediana
24         int n = numeros.size();
25         if (n % 2 == 0) {
26             return (numeros.get(n / 2) + numeros.get(n / 2 - 1)) / 2;
27         } else {
28             return numeros.get(n / 2);
29         }
30     }
31
32     public double moda(ArrayList<Object> valores) {
33         // HashMap para mantener el conteo de la frecuencia de cada valor
34         HashMap<Double, Integer> freqMap = new HashMap<>();
35         // Recorre todos los elementos en el ArrayList
36         for (Object valor : valores) {
37             // Convierte cada objeto a Double y actualiza su frecuencia en el mapa
38             double num = Double.parseDouble(valor.toString());
39             freqMap.put(num, freqMap.getOrDefault(num, 0) + 1);
40         }
41         // Encuentra el valor con la mayor frecuencia
42         double mode = Double.parseDouble(valores.get(0).toString());
43         int maxCount = 0;
44         for (Map.Entry<Double, Integer> entry : freqMap.entrySet()) {
45             if (entry.getValue() > maxCount) {
46                 maxCount = entry.getValue();
47                 mode = entry.getKey();
48             }
49         }
50         return mode;
51     }
52
53     public double varianza(ArrayList<Object> valores) {
54         // Calcula la media de los valores en el ArrayList
55         double media = media(valores);
56         double suma = 0;
57         // Recorre todos los elementos en el ArrayList
58         for (Object valor : valores) {
59             // Calcula la diferencia al cuadrado entre el valor y la media y suma los resultados
60             suma += Math.pow(Double.parseDouble(valor.toString()) - media, 2);
61         }
62         // Retorna la varianza
63         return suma / valores.size();
64     }
65
66     public double maximo(ArrayList<Object> valores) {
67         // Utiliza streams para encontrar el valor máximo en el ArrayList
68         return valores.stream()
69             .mapToDouble(value -> Double.parseDouble(value.toString()))//conviernte a double
70             .max()//busca el maximo
71             .orElse(0.0); // Si el ArrayList está vacío, devuelve 0.0
72     }
73
74     public double minimo(ArrayList<Object> valores) {
75         // Utiliza streams para encontrar el valor mínimo en el ArrayList
76         return valores.stream()
77             .mapToDouble(value -> Double.parseDouble(value.toString())) //convierte a double
78             .min()//busca el minimo
79             .orElse(0.0); // Si el ArrayList está vacío, devuelve 0.0
80     }
81 }
82 
```

Figura 11 Clase OperacionesEstadísticas

Fuente: Elaboración Propia

Glosario

- **Analizador léxico:** Lee el código fuente línea por línea, carácter por carácter de izquierda a derecha y los agrupa en componentes léxicos o tokens.
- **Lenguaje:** Conjunto de cadenas o palabras de un alfabeto.
- **Lexema:** Secuencias válidas para un token que cumple el patrón.
- **Patrón:** Describe las reglas que describen la validez del token.
- **Símbolo:** Son letras, dígitos, caracteres del lenguaje.
- **Token:** Secuencia de caracteres que tienen un significado lógico colectivo.

Conclusiones

- Se debe de tener clara la idea de cómo estructurar los métodos para que estos no retornen datos incorrectos o erróneos.
- La recursividad es muy compleja por lo que se debe de tener cuidado a la hora de implementarla.
- Al analizar elemento por elemento se debe de tener claro que se está haciendo para no perderse, se debe de establecer la posición de cada elemento .
- El manejo de archivos y un amplio conocimiento sobre entorno grafico facilitan el manejo del programa.