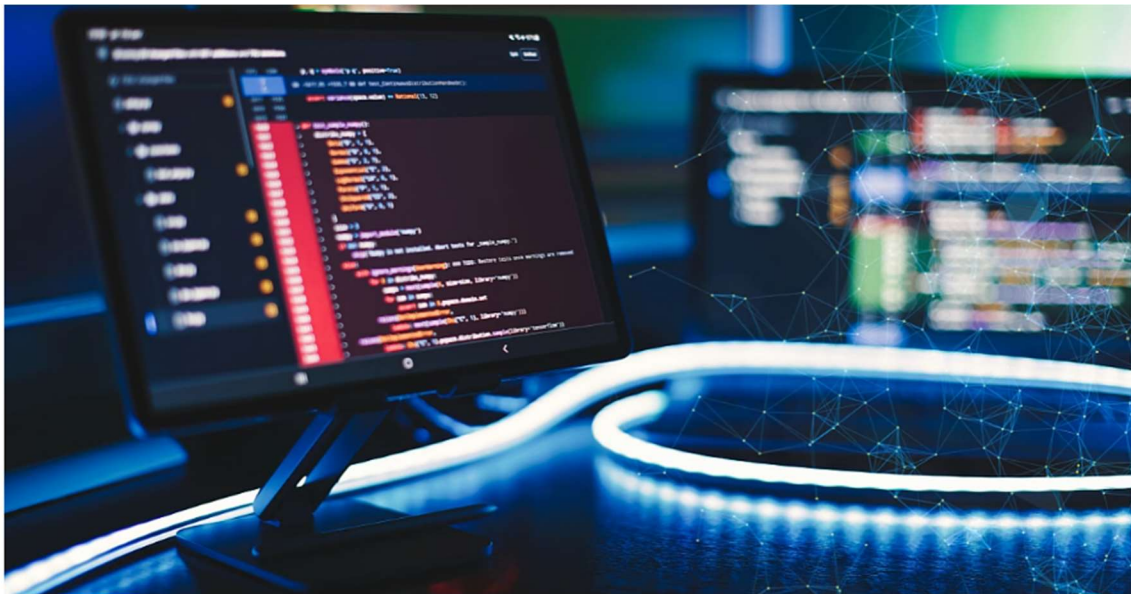


# Manual Técnico



**Organización de Lenguajes y Compiladores 1**

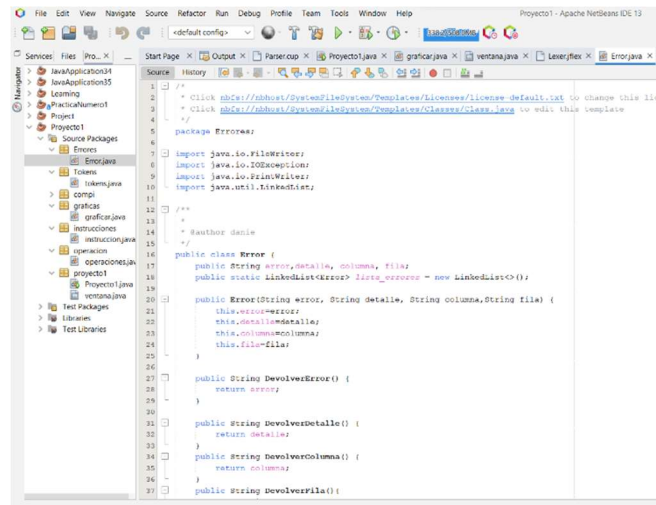
**Francisco Daniel Peruch de León**

**Carné: 202100639**

- **Paquete Errores:**

En este paquete se creó la clase "Error" la cual está destinada para manejar la información relacionada a los errores encontrados en el análisis. A continuación, se detalla los métodos y atributos relacionados a esta clase:

- ✓ Se crearon los atributos: error, detalla, columna y fila los cuales son de tipo string. Estos fueron creados para almacenar la información relacionada a cada carácter que no fue reconocido por el analizador.
- ✓ Se creo una LinkedList que contendrá objetos de la clase Error, el cual fue nombrado lista\_errores. Este fue declarado como static para que de esta manera se pueda acceder desde diferentes partes del proyecto.
- ✓ public Error (String error, String detalle, String columna, String fila): Este es el constructor de la clase Error, por medio de este se crearán diferentes objetos que contendrá la información relacionada a el error encontrado en el análisis.
- ✓ public String DevolverError (): Por medio de este método se puede obtener el carácter que fue encontrado como error en el análisis.
- ✓ public String DevolverDetalle (): Por medio de este método se puede obtener el detalle del error encontrado.
- ✓ public String DevolverColumna (): Por medio de este método se puede obtener la columna donde se encontró el error.
- ✓ public String DevolverFila (): Por medio de este método se puede obtener la fila donde se encontró el error.
- ✓ public static void GuardarError (String error, String detalle, String columna, String fila): Este método es utilizado para guardar la información referente al error encontrado, debido a que este método es invocado desde el archivo. jflex, de donde se obtendrá la información del error, se declaró un método static para que de esta manera sea más sencillo llamarlo.
- ✓ public static void MostrarErrores (): Este método tiene la función de crear un archivo html, con la información referente a los errores encontrados durante el análisis. Este fue declarado como un método static para que sea más sencillo llamarlo desde la clase ventana, que es en donde se maneja la interfaz gráfica de la aplicación.

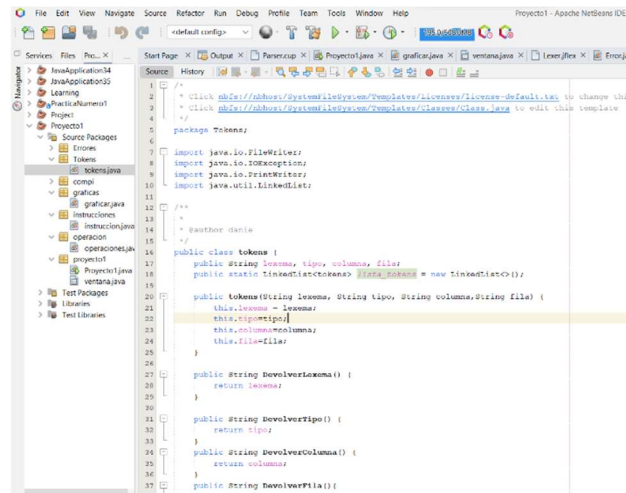


- **Paquete Tokens:**

En este paquete se creó la clase tokens, la cual tiene diferentes métodos y atributos que serán utilizados para guardar la información de los tokens encontrados en el análisis. A continuación, se muestra el detalle de los métodos y atributos correspondientes a esta clase:

- ✓ Se crearon los siguientes atributos públicos: lexema, tipo, columna, fila. Estos servirán para guardar información de los lexemas encontrados en el análisis.
- ✓ public tokens (String lexema, String tipo, String columna, String fila): Este es el constructor de la clase tokens, el cual se usará para crear diferentes objetos para manejar la información relacionada a los tokens.
- ✓ Se creó una LinkedList que contendrá objetos de la clase tokens, el cual fue nombrado lista\_tokens. Este fue declarado como static para que de esta manera se pueda acceder desde diferentes partes del proyecto.
- ✓ public tokens (String lexema, String tipo, String columna, String fila): Este es el constructor de la clase tokens, por medio de este se crearán diferentes objetos que contendrán la información relacionada al token encontrado en el análisis.
- ✓ public String DevolverLexema (): Por medio de este método se puede obtener el lexema encontrado en el análisis.
- ✓ public String DevolverTipo (): Por medio de este método se puede obtener el tipo de token que se encontró en el análisis.
- ✓ public String DevolverColumna (): Por medio de este método se puede obtener la columna donde se encontró el token.

- ✓ `public String DevolverFila ()`: Por medio de este método se puede obtener la fila donde se encontró el token.
- ✓ `public static void GuardarTokes (String lexema, String tipo, String column, String fila)`: Este método es utilizado para guardar la información referente al token encontrado, debido a que este método es invocado desde el archivo. `jflex`, de donde se obtendrá la información de los tokens, se declaró como un método static para que de esta manera sea más sencillo llamarlo.
- ✓ `public static void MostrarTokens ()`: Este método tiene la función de crear un archivo html, con la información referente a los tokens encontrados durante el análisis. Este fue declarado como un método static para que sea más sencillo llamarlo desde la clase `ventana`, que es en donde se maneja la interfaz gráfica de la aplicación.



- **Paquete graficas:**

En este paquete se creo una clase llamada `graficar`, la cual contiene diferentes método y atributos que se utilizaron para manejar la información para crear las gráficas, además de ciertos métodos que se usaron para mostrar la información en la consola de la aplicación.

A continuación, se detallan los atributos y métodos correspondiente a esta clase. Los atributos fueron declarados como static para que fuera más sencillo el llamar estos atributos en diferentes partes del proyecto donde se necesitara usarlos:

- ✓ `public static String titulobarras`: Este atributo se utiliza para guardar la cadena correspondiente a el titulo de la grafica de barras a crear.
- ✓ `public static String tituloxbarras`: Este atributo se utiliza para guardar la cadena referente al título en el eje x de la gráfica.

- ✓ `public static String tituloYbarras`: Este atributo se utiliza para guardar la cadena referente al título que se mostrara en el eje y de la gráfica.
- ✓ `public static LinkedList<String> ejeYbarras`: Este `LinkedList` se utilizó para guardar los valores correspondientes al eje y, los cuales se obtuvieron al realizar el análisis.
- ✓ `public static LinkedList<String> ejeXbarras`: Este `LinkedList` se utilizó para guardar los valores correspondientes al eje x, los cuales se obtuvieron al realizar el análisis.
- ✓ `public static String tituloPie`: Este atributo se utilizó para guardar el título de la gráfica de Pie que se reconoció en el análisis.
- ✓ `public static LinkedList<String> valoresPie` = Por medio de este `LinkedList` se puede almacenar cada uno de los valores que se reconocieron durante el análisis, referentes a los valores.
- ✓ `public static LinkedList<String> labelPie`: Por medio de este `LinkedList` se puede almacenar cada una de las etiquetas que se reconocieron durante el análisis, para esta gráfica.
- ✓ `public static String tituloLineal`: En este se guarda la cadena correspondiente a el título de la gráfica de línea.
- ✓ `public static String tituloXlineal`: En este se guarda la cadena que hace referencia a el título en el eje x.
- ✓ `public static String tituloYlineal`: En este se guarda la cadena correspondiente al título en y.
- ✓ `public static LinkedList<String> ejeYlineal`: Por medio de esta `LinkedList` se guarda cada uno de los valores reconocidos en el análisis, los cuales corresponden a los valores en el eje y.
- ✓ `public static LinkedList<String> ejeXlineal`: Por medio de esta `LinkedList` se almacena cada uno de los valores correspondiente al eje x en la gráfica.
- ✓ `public static LinkedList<Double> valoresHistograma`: En esta `LinkedList` se almacena los valores utilizados para el análisis.
- ✓ `public static LinkedList<Double> frecuenciaHistograma`: En esta `LinkedList` se almacena los valores referentes a la frecuencia de cada dato reconocido en el análisis.
- ✓ `public static LinkedList<Double> frecuenciaAcumulada`: En esta `LinkedList` se almacena los valores referentes a la frecuencia acumulada de cada dato reconocido en el análisis.
- ✓ `public static LinkedList<Integer> frecuenciaRelativa`: En este `LinkedList` se almacena cada valor relacionado a la frecuencia relativa de los valores reconocidos durante el análisis.
- ✓ `public static String tituloHistograma`: En este atributo se guarda la cadena correspondiente al título que se mostrara en la gráfica del histograma.
- ✓ `public static String cadenaAnalizar`: Este atributo es utilizado para almacenar el texto que se obtuvo del Área de Texto para luego realizar el análisis correspondiente.

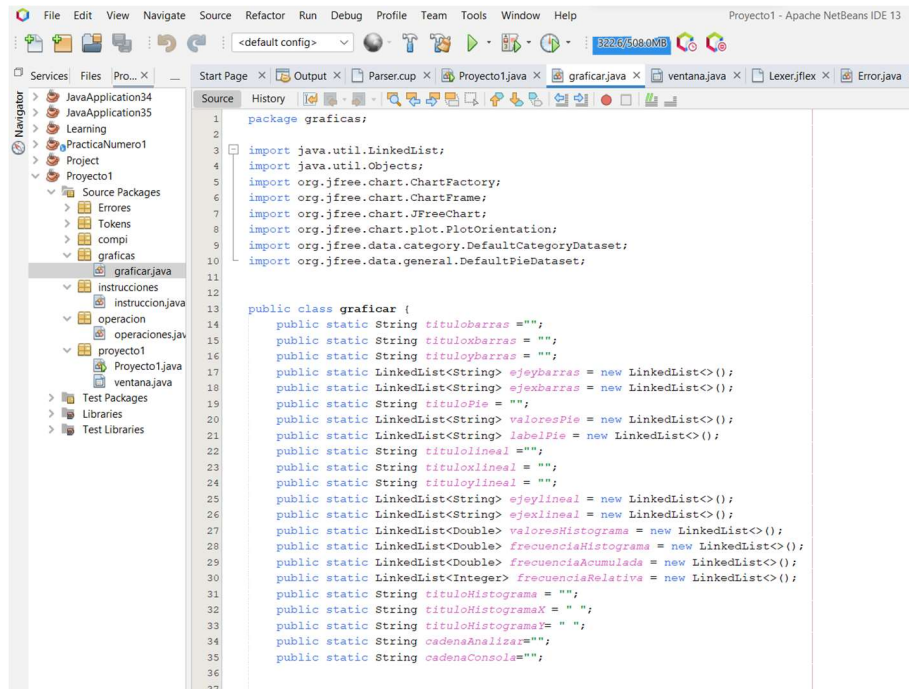
- ✓ `public static String cadenaConsola`: Este atributo es utilizado para guardar la cadena que será mostrada en la consola de la aplicación.
- ✓ `public static void barras`: Este método recibe como parámetro el título de la gráfica de barras, el título en el eje x, el título en el eje y, un `LinkedList` con los valores para el eje x y otro `LinkedList` para los valores en el eje y usando estos parámetros se genera la gráfica de barras en esta función.
- ✓ `public static void histograma`: Este método recibe como parámetro los valores del título de la gráfica del histograma, el título para el eje x, el título para el eje y dos `LinkedList` que almacenan los valores reconocidos en el análisis y la frecuencia de estos, las cuales fueron encontradas durante el análisis.
- ✓ `public static void Lineal`: Este método recibe como parámetro los valores del título de la gráfica de línea, el título para el eje x, el título para el eje y dos `LinkedList` que almacenan los valores reconocidos en los análisis correspondientes a los ejes de la gráfica.
- ✓ `public static void Pie`: Este método recibe como parámetro los valores del título de la gráfica de pie además de dos `LinkedList` que almacenan los valores reconocidos en el análisis y las etiquetas de estas.
- ✓ `public static void AgregarTituloBarras`: Este método toma como parámetro un valor de tipo `String` y se lo asigna a el atributo `titulobarras`, el cual se utilizará para generar la gráfica.
- ✓ `public static void AgregarTituloXBarras`: Este método toma como parámetro un valor de tipo `String` y se lo asigna a el atributo `tituloxbarras`, el cual se utilizará para generar la gráfica.
- ✓ `public static void AgregarTituloYBarras`: Este método toma como parámetro un valor de tipo `String` y se lo asigna a el atributo `tituloybarras`, el cual se utilizará para generar la gráfica.
- ✓ `public static void AgregarEjeYBarras`: Este método toma como parámetro una `LinkedList` la cual asigna a el atributo `ejeybarras`.
- ✓ `public static void AgregarEjeXBarras`: Este método toma como parámetro una `LinkedList` la cual asigna a el atributo `ejexbarras`.
- ✓ `public static void AgregarTituloPie`: Este método toma como parámetro un valor de tipo `String` y se lo asigna a el atributo `tituloPie`, el cual se utilizará para generar la gráfica.
- ✓ `public static void AgregarValoresPie`: Este método toma como parámetro una `LinkedList` la cual asigna a el atributo `valoresPie`.
- ✓ `public static void AgregarLabelsPie`: Este método toma como parámetro una `LinkedList` la cual asigna a el atributo `labelPie`.
- ✓ `public static void AgregarTituloLineal`: Este método toma como parámetro un valor de tipo `String` y se lo asigna a el atributo `titulolineal`, el cual se utilizará para generar la gráfica.

- ✓ `public static void AgregarTituloXLineal`: Este método toma como parámetro un valor de tipo `String` y se lo asigna a el atributo `tituloxlineal`, el cual se utilizará para generar la gráfica.
- ✓ `public static void AgregarTituloYLineal`: Este método toma como parámetro un valor de tipo `String` y se lo asigna a el atributo `tituloylineal`, el cual se utilizará para generar la gráfica.
- ✓ `public static void AgregarEjeYLineal`: Este método toma como parámetro una `LinkedList` la cual asigna a el atributo `ejeyleal`.
- ✓ `public static void AgregarEjeXLineal`: Este método toma como parámetro una `LinkedList` la cual asigna a el atributo `ejexlineal`.
- ✓ `public static void AgregarValoresHistograma`: Este método toma como parámetro una `LinkedList` la cual asigna a el atributo `valoresHistograma`.
- ✓ `public static void AgregarFrecuenciaHistograma`: Este método toma como parámetro una `LinkedList` la cual asigna a el atributo `frecuenciaHistograma`.
- ✓ `public static void AgregarFrecuenciaAcumuladaHistograma`: Este método toma como parámetro una `LinkedList` la cual asigna a el atributo `frecuenciaAcumulada`.
- ✓ `public static void AgregarFrecuenciaRelativaHistograma`: Este método toma como parámetro una `LinkedList` la cual asigna a el atributo `frecuenciaRelativa`.
- ✓ `public static LinkedList<Double> crearValores`: Este método recibe como parámetro un `LinkedList` la cual contiene todos los valores reconocidos en el análisis, en este `LinkedList` aparecen los datos repetidos que se reconocieron. Entonces usando este `LinkedList` se hace un `for` para castear cada uno de esto valores para que sean de tipo `double` y luego se asignan a una nueva `LinkedList` llamada `lista`, además se crea una nueva `LinkedList` llamada `lista_valores`, la cual será la que almacenara los valores no repetidos. De esta manera se recorre la `lista` y se revisa si ya esta contenida en la `lista_valores` y sí no se agrega. Al final esta `lista_valores` es la que retorna el método.
- ✓ `public static LinkedList<Double> frecuencia`: Este método recibe como parámetro dos `LinkedList`, una que contiene los valores encontrados en el análisis, los cuales aparecen repetido si es el caso y otra que contiene los valores reconocidos en el análisis los cuales no están repetidos. Usando la `lista` que contiene los datos no repetidos, se toma un valor y se va revisando cuantas veces aparece en la otra `lista` y esa cantidad se agrega en una nueva `LinkedList` la cual es la que devuelve le método, la cual contendrá la frecuencia de cada valor.
- ✓ `public static LinkedList<Double> frecuenciaAcumuladaHistograma`: Este método recibe como parámetro una `LinkedList` la cual contiene los valores de la frecuencia de cada dato reconocido en el análisis, luego se crea una variable `suma`, la cual se inicializa con un valor de cero, se recorre la `lista` y se va sumando cada valor que contiene la `lista` recibida por parámetro a la

variable suma y luego se guarda en una nueva LinkedList, la cual al finalizar el método contendrá los valores de la frecuencia acumulada para luego ser devuelta por este método.

- ✓ `public static LinkedList<Integer> frecuenciaRelativaHistograma`: Este método se recibe como parámetro dos LinkedList, una que contiene la frecuencia de cada dato reconocido en el análisis y en el otro los valores reconocidos desde el texto que se analizó en el cual se encuentran los valores repetidos, la finalidad de este último es mostrar cuantos datos se reconocieron en el análisis en primera instancia, y usando ese dato se puede calcular la frecuencia relativa. Esto se logra tomando los valores de cada frecuencia y dividirlo dentro de la cantidad de datos analizados. Para al final multiplicar por 100. Luego se castea a un int este dato correspondiente a la frecuencia relativa y se agrega a una LinkedList que almacena integers.
- ✓ `public static void AgregarTituloHistograma`: Este método recibe como parámetro un string el cual asigna a el atributo tituloHistograma.
- ✓ `public static void CrearAnalizadores`: Este método se utiliza para crear los analizadores cada vez que se tiene un nuevo cambio en los archivos. jflex o cup. Así de esta manera se puede actualizar los cambios.
- ✓ `public static void reporteHistograma`: Este método recibe cuatro LinkedList los cuales contienen los datos de: valores reconocidos, frecuencia, frecuencia acumulada y frecuencia relativa de estos valores. Usando estos datos se va creando una cadena de texto que contenga toda la información necesaria para ser trasladada a la consola de la aplicación.
- ✓ `public static void crearCadenaConsola`: En este método se recibe como parámetro un string el cual se va agregando a una variable static llamada cadenaConsola que va almacenando toda la información que se desea mostrar en la consola.
- ✓ `public static void crearCadenaLista`: Este método recibe una LinkedList que almacena valores tipo String, los cuales se van agregando a una variable string static llamada cadenaConsola, que es la que tendrá toda la información que se necesita mostrar en la consola de la aplicación.





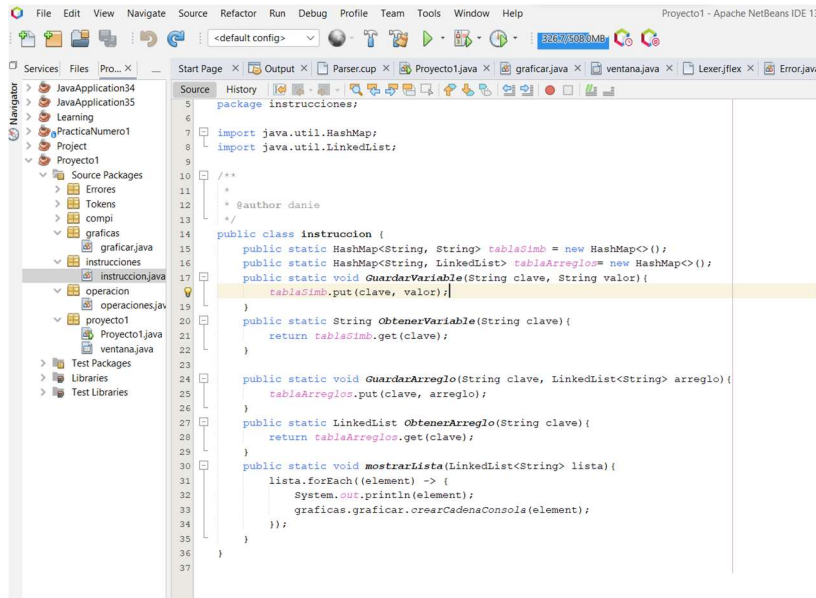
- **Paquete instrucciones**

En este paquete se creó una clase llamada Instruccion, en esta clase se maneja principalmente el tema de guardar las variables y arreglos en HashMaps. En esta clase se hizo uso de los siguientes atributos y métodos.

- ✓ **tablaSimb:** Este es un HashMap que contiene las variables que se reconocieron durante el análisis.
- ✓ **tablaArreglos:** Este es un HashMap que contiene los arreglos que se reconocieron durante el análisis.
- ✓ **public static void GuardarVariable:** Este método recibe como parámetro dos valores tipo String, uno correspondiente al ID de la variable y otro correspondiente al valor de la variable. Estos datos son ingresados a el atributo tablaSimb, usando el valor correspondiente al ID de la variable como la clave y el valor de la variable como valor.
- ✓ **public static String ObtenerVariable:** Este método recibe como parámetro un valor tipo String, el cual es el correspondiente al ID de la variable que se desea obtener su valor. Este valor que se recibe como parámetro se usa como clave para buscar en el atributo tablaSimb el valor que se desea.
- ✓ **public static void GuardarArreglo:** Este método recibe como parámetros un valor tipo String que corresponde al ID del arreglo y

una LinkedList que contiene los valores que guarda el arreglo. Se utiliza el id del arreglo como la clave y la LinkedList como el valor al momento de agregarlo en la tablaArreglos

- ✓ public static LinkedList ObtenerArreglo: Este método recibe como parámetro el id del arreglo para buscar dentro de tablaArreglos la LinkedList correspondiente a ese id, para devolverla.
- ✓ public static void mostrarLista: Este método se utiliza para mostrar los datos que guarda un arreglo, en la consola de la aplicación.



```
5 package instrucciones;
6
7 import java.util.HashMap;
8 import java.util.LinkedList;
9
10 /**
11  *
12  * @author daniel
13  */
14 public class Instruccion {
15     public static HashMap<String, String> tablaSimb = new HashMap<>();
16     public static HashMap<String, LinkedList> tablaArreglos = new HashMap<>();
17     public static void GuardarVariable(String clave, String valor) {
18         tablaSimb.put(clave, valor);
19     }
20     public static String ObtenerVariable(String clave) {
21         return tablaSimb.get(clave);
22     }
23     public static void GuardarArreglo(String clave, LinkedList<String> arreglo) {
24         tablaArreglos.put(clave, arreglo);
25     }
26     public static LinkedList ObtenerArreglo(String clave) {
27         return tablaArreglos.get(clave);
28     }
29     public static void mostrarLista(LinkedList<String> lista) {
30         lista.forEach((element) -> {
31             System.out.println(element);
32             graficas.graficar.crearCadenaConsola(element);
33         });
34     }
35 }
36
37 }
```

- **Paquete operacion:**

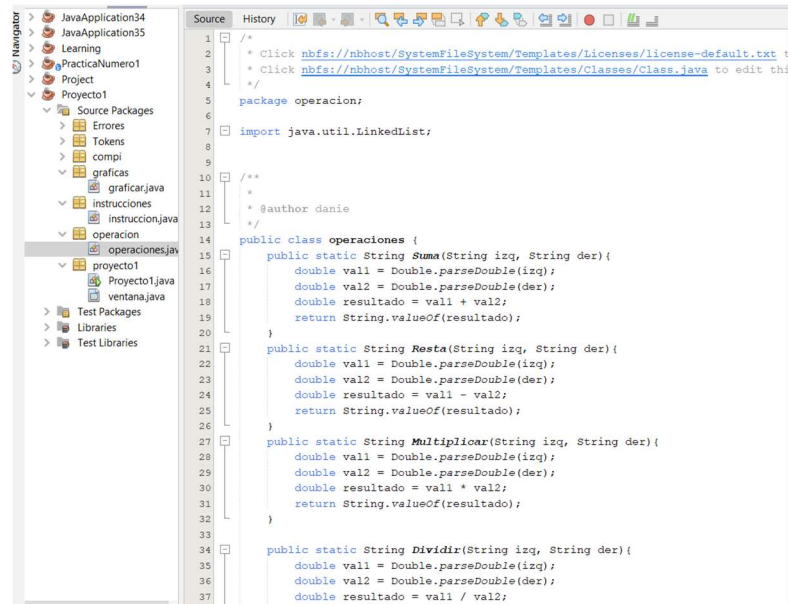
En este paquete se creo una clase llamada operaciones en donde se realizan principalmente las operaciones aritméticas y estadísticas que se necesiten durante el análisis. A continuación, se detallan los atributos y métodos empleados en esta clase:

- ✓ public static String Suma (String izq, String der): Este método recibe como parámetro dos valores de tipo String los cuales se castean a valores double, para luego devolver la suma de estos dos valores.
- ✓ public static String Resta (String izq, String der): Este método recibe como parámetro dos valores de tipo String los cuales se castean a valores double, para luego devolver la resta de estos dos valores.
- ✓ public static String Multiplicar (String izq, String der): Este método recibe como parámetro dos valores de tipo String los cuales se castean a valores double, para luego devolver la multiplicación de estos dos valores.

- ✓ `public static String Dividir (String izq, String der)`: Este método recibe como parámetro dos valores de tipo `String` los cuales se castean a valores `double`, para luego devolver la división de estos dos valores.
- ✓ `public static String Modulo (String izq, String der)`: Este método recibe como parámetro dos valores de tipo `String` los cuales se castean a valores `double`, para luego devolver el módulo de estos dos valores.
- ✓ `public static String Media (LinkedList<String> listaDatos)`: Este método recibe como parámetro una `LinkedList` que contiene los datos de los cuales se quiere obtener la media, para esto se crea un arreglo el cual contendrá los valores de esta `LinkedList`, pero dichos valores serán de tipo `double`, para luego utilizar una variable `double` llamada `total`, que se inicializa en cero. Cada dato del arreglo se ira sumando a la variable `total`. Para obtener la media se toma esa variable `total` y se divide dentro del tamaño de la `LinkedList` que se obtuvo por parámetro.
- ✓ `public static String Mediana (LinkedList<String> listaDatos)`: Este método recibe como parámetro una `LinkedList` con los datos a los cuales se desea conseguir la mediana. Primero se crea un arreglo del mismo tamaño de la `LinkedList` que se recibe por parámetro. Luego se debe realizar un ordenamiento en ese arreglo, para ello se utiliza el ordenamiento de burbuja. Teniendo el arreglo ordenado, se procede a verificar si el arreglo es de tamaño par o impar. Si es par, se obtiene el valor en el índice igual al tamaño del arreglo dividido dos y el valor en el índice igual al tamaño del arreglo dividido dos menos uno. Luego se hace un promedio de estos datos y esa sería la mediana, mientras que si es par solo se obtiene el valor del índice de en medio y esa sería la mediana.
- ✓ `public static String Moda (LinkedList<String> listaDatos)`: En este método se recibe como parámetro una `LinkedList` con los datos a los que se necesita obtener la Moda, para ello se crea un arreglo del mismo tamaño que la `LinkedList`, este arreglo contendrá los mismos valores que la `LinkedList`, pero serán de tipo `double`. Usando el arreglo de `doubles`, se va tomando cada valor y se revisa cuantas veces se repite dentro del arreglo. Para ello se declaran tres variables `double` las cuales son las siguientes: `temporal`, `moda`, `contador_dato`. La variable `temporal` sirve para tomar el valor del dato que se va a analizar cuantas veces se repite en el arreglo, la variable `moda` sirve para tomar el valor del dato que más se ha repetido durante la iteración y `contador_dato` es la que llevara el registro de la cantidad de veces que más se ha repetido un dato durante la iteración. Cada vez que un valor se encuentre en el arreglo se le sumara el valor de uno a una variable `contador` y se

verificara si esa variable se ha repetido más veces que la que lleva más veces repetida hasta ese momento de la iteración. Si se verifica que sí, esa variable pasa a ser la nueva moda, al final de la iteración el método devuelve el valor de moda.

- ✓ `public static String Varianza (LinkedList<String> lista_datos)`: Este método recibe como parámetro una `LinkedList` que contiene los datos de los cuales se requiere obtener la varianza, para ellos se procede a pasar cada uno de los datos de la `LinkedList` a un arreglo, solo que los datos se castearan a `double` para agregarse al arreglo, luego se aplicara la formula para calcular la varianza.
- ✓ `public static String Menor (LinkedList<String> listaDatos)`: Este método recibe como parámetro una `LinkedList` que contiene los datos de los cuales se requiere obtener el número más pequeño, para ello se va comparando cada dato para reconocer cual es el más pequeño.
- ✓ `public static String Mayor (LinkedList<String> listaDatos)`: Este método recibe como parámetro una `LinkedList` que contiene los datos de los cuales se requiere obtener el número más grande, para ello se va comparando cada dato para reconocer cual es el más grande.



The screenshot shows an IDE with a project structure on the left and a source code editor on the right. The project structure includes a package named 'operacion' containing a file 'operaciones.java'. The source code in 'operaciones.java' defines a class 'operaciones' with four static methods: 'Suma', 'Resta', 'Multiplicar', and 'Dividir'. Each method takes two strings as input, converts them to doubles, performs the respective arithmetic operation, and returns the result as a string.

```
1  /*  
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to  
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this  
4  */  
5  package operacion;  
6  
7  import java.util.LinkedList;  
8  
9  
10 /**  
11  *  
12  * @author daniel  
13  */  
14 public class operaciones {  
15     public static String Suma(String izq, String der){  
16         double val1 = Double.parseDouble(izq);  
17         double val2 = Double.parseDouble(der);  
18         double resultado = val1 + val2;  
19         return String.valueOf(resultado);  
20     }  
21     public static String Resta(String izq, String der){  
22         double val1 = Double.parseDouble(izq);  
23         double val2 = Double.parseDouble(der);  
24         double resultado = val1 - val2;  
25         return String.valueOf(resultado);  
26     }  
27     public static String Multiplicar(String izq, String der){  
28         double val1 = Double.parseDouble(izq);  
29         double val2 = Double.parseDouble(der);  
30         double resultado = val1 * val2;  
31         return String.valueOf(resultado);  
32     }  
33  
34     public static String Dividir(String izq, String der){  
35         double val1 = Double.parseDouble(izq);  
36         double val2 = Double.parseDouble(der);  
37         double resultado = val1 / val2;  
38     }  
39 }
```