Universidad San Carlos de Guatemala

Facultad de Ingeniería

Organización De Lenguajes Y Compiladores, Sección C

Ing. Kevin Adiel Lajpop Ajpacaja

Auxiliar: Carlos Daniel Acabal Pérez

Primer Semestre 2024

Tema:

Manuel Técnico – DataForge

Nombre: Carlos Manuel Lima y Lima

Registro Académico: 202201524

CUI: 3009368850101

Guatemala, 10 de marzo del 2024.

APPSTATE

La clase AppState en el paquete dataforge es un contenedor centralizado para gestionar y compartir información relevante en la aplicación. Se utiliza para almacenar datos importantes como tokens, errores léxicos, errores sintácticos y símbolos nodos. A continuación, se describen los elementos clave de la clase:

1. public static LinkedList<Token> listaToken = new LinkedList<>();

- Descripción: Almacena una lista enlazada de objetos Token.
- Uso: Se utiliza para registrar tokens generados durante el análisis del código fuente.

2. public static LinkedList<ErrorLexico> listaErrorLexico = new LinkedList<>();

- Descripción: Contiene una lista enlazada de objetos ErrorLexico.
- Uso: Almacena errores léxicos identificados durante el análisis del código fuente.

3. public static LinkedList<ErrorSintactico> listaErrorSintactico = new LinkedList<>();

- Descripción: Almacena una lista enlazada de objetos ErrorSintactico.
- Uso: Registra errores sintácticos detectados durante el análisis del código fuente.

4. public static LinkedList<SimboloNodo> tablaSimbolo = new LinkedList<>();

- Descripción: Guarda una lista enlazada de objetos SimboloNodo.
- Uso: Utilizado para mantener una tabla de símbolos, que podría representar identificadores y su información asociada.

Uso de la Clase:

- **Acceso Compartido:** Al ser declaradas como static, estas variables pertenecen a la clase en lugar de a instancias individuales, permitiendo que sean compartidas entre diferentes partes del código.
- **Centralización de Datos:** AppState actúa como un punto centralizado para acceder y gestionar datos críticos de la aplicación, facilitando la coherencia y la facilidad de mantenimiento.
- **Facilita la Depuración:** Almacenar información importante en estas listas facilita el seguimiento y la depuración de problemas, ya que los diferentes componentes de la aplicación pueden acceder y modificar estos datos compartidos de manera controlada.

```java
package dataforge;
import java.util.LinkedList;
import Instrucciones.SimboloNodo;
/**
 *
 * @author manuel
 */
public class AppState {
    public static LinkedList<Token> listaToken = new LinkedList<>();
    public static LinkedList<ErrorLexico> listaErrorLexico = new LinkedList<>();
    public static LinkedList<ErrorSintactico> listaErrorSintactico = new LinkedList<>();
    public static LinkedList<SimboloNodo> tablaSimbolo = new LinkedList<>();
}
```

VENTANA PRINCIPAL

Nuevo Archivo

1. Selector de Archivos (JFileChooser):

Se crea una instancia del objeto JFileChooser, proporcionando una interfaz gráfica para que el usuario seleccione la ubicación y el nombre del nuevo archivo. El diálogo se personaliza para mostrar el título "Nuevo Archivo" y filtrar solo archivos con extensión DF.

2. Mostrar el Diálogo de Guardar Archivo:

Se muestra el diálogo de selección de archivo, y la respuesta del usuario se almacena en la variable seleccion. Si el usuario elige una ubicación y proporciona un nombre de archivo, se procede a crear el nuevo archivo.

3. Verificación y Creación del Archivo:

Se recupera el archivo seleccionado y se verifica si su nombre termina con la extensión DF. Si no es así, se ajusta el nombre del archivo para incluir la extensión correcta. Luego, se intenta crear el archivo y se escribe un contenido vacío en él.

4. Creación de una Pestaña en jTabbedPane1:

Si la creación del archivo es exitosa, se crea un área de texto (JTextArea) y se le asigna un tipo de letra específico. Luego, se coloca esta área de texto en un panel con barra de desplazamiento (JScrollPane). Finalmente, se agrega esta pestaña al componente jTabbedPane1, que probablemente sea un contenedor de pestañas en la interfaz de usuario.

```
private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle( dialogTitle: "Nuevo Archivo");
    fileChooser.setFileFilter(new FileNameExtensionFilter( description: "Archivos DF (*.df)",  extensions: "df"));

    int seleccion = fileChooser.showSaveDialog( parent: this);

    if (seleccion == JFileChooser.APPROVE_OPTION) {
        File archivo = fileChooser.getSelectedFile();
        if (!archivo.getName().toLowerCase().endsWith( suffix: ".df")) {
            archivo = new File(archivo.getAbsolutePath() + ".df");
        }
        try {
            if (archivo.createNewFile()) {
                try (FileWriter writer = new FileWriter( file: archivo)) {
                    writer.write( str: "");
                }
                JTextArea textArea = new JTextArea();
                textArea.setFont(new Font( name: "Arial Rounded MT Bold",  style: Font.PLAIN,  size: 10));
                JScrollPane scrollPane = new JScrollPane( view: textArea);
                jTabbedPane1.addTab( title: archivo.getName(),  component: scrollPane);
                JOptionPane.showMessageDialog( parentComponent: this,  message: "Archivo Y Pestaña Creada Correctamente",  title: "Nuevo Archivo",  messageType: JOptionPane.INFORMATION_MESSAGE
            }
        } catch (IOException e) {
            JOptionPane.showMessageDialog( parentComponent: this, "Error: " + e,  title: "Error Nuevo Archivo",  messageType: JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Abrir Archivo

1. Selector de Archivos (JFileChooser):

Se instancia un objeto JFileChooser para proporcionar una interfaz gráfica que permita al usuario seleccionar un archivo. El diálogo se configura con el título "Abrir Archivo" y se filtra para mostrar solo archivos con extensión DF.

2. Mostrar el Diálogo de Abrir Archivo:

Se muestra el diálogo de selección de archivo, y la respuesta del usuario se almacena en la variable seleccion. Si elige un archivo, se procede a abrirlo.

3. Lectura del Contenido del Archivo:

Se obtiene el archivo seleccionado y se utiliza un BufferedReader para leer su contenido. Este contenido se acumula en un StringBuilder mientras se lee línea por línea desde el archivo.

4. Creación de una Pestaña en jTabbedPane1:

- Se crea un área de texto (JTextArea) y se le asigna el contenido leído del archivo.
- Se establece un tipo de letra específico para el área de texto.
- Se coloca el área de texto en un panel con barra de desplazamiento (JScrollPane).
- Se agrega esta pestaña al componente jTabbedPane1, que probablemente sea un contenedor de pestañas en la interfaz de usuario.

```
private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle( dialogTitle: "Abrir Archivo");
    fileChooser.setFileFilter(new FileNameExtensionFilter( description: "Archivos DF (*.df)",  extensions: "df"));

    int seleccion = fileChooser.showOpenDialog( parent: this);

    if (seleccion  == JFileChooser.APPROVE_OPTION) {
        File archivo = fileChooser.getSelectedFile();

        try (BufferedReader reader = new BufferedReader(new FileReader( file:archivo))) {
            StringBuilder contenido = new StringBuilder();
            String linea;
            while ((linea = reader.readLine()) != null) {
                contenido.append( str: linea).append( str: "\n");
            }
            JTextArea textArea = new JTextArea( text: contenido.toString());
            textArea.setFont(new Font( name: "Arial Rounded MT Bold",  style: Font.PLAIN,  size: 10));
            JScrollPane scrollPane = new JScrollPane( view: textArea);
            jTabbedPane1.addTab( title: archivo.getName(),  component: scrollPane);
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Archivo Abierto Correctamente",  title: "Nuevo Archivo",  messageType: JOptionPane.INFORMATION_MESSAGE);
        } catch (IOException e) {
            JOptionPane.showMessageDialog( parentComponent: this, "Error: " + e,  title: "Error Abrir Archivo",  messageType: JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Guardar

1. Selector de Archivos (JFileChooser):

Se instancia un objeto JFileChooser para proporcionar una interfaz gráfica que permita al usuario seleccionar la ubicación y el nombre para guardar el archivo. El diálogo se configura con el título "Guardar" y se filtra para mostrar solo archivos con extensión DF.

2. Obtención de la Pestaña Seleccionada:

Se obtiene el índice de la pestaña actualmente seleccionada en el componente jTabbedPane1.

3. Verificación de Pestaña Seleccionada:

Si no hay ninguna pestaña abierta (PestanaSeleccionada == -1), se muestra un mensaje de error indicando que no hay pestañas abiertas y se interrumpe el proceso de guardar.

4. Mostrar el Diálogo de Guardar Archivo:

Se muestra el diálogo de selección de archivo para guardar, y la respuesta del usuario se almacena en la variable returnValue. Si elige una ubicación y proporciona un nombre de archivo, se procede a guardar.

5. Obtención del Contenido del Área de Texto:

Se obtiene el contenido del área de texto asociada a la pestaña seleccionada.

6. Escritura del Contenido en el Archivo:

Se obtiene el archivo seleccionado y se utiliza un BufferedWriter para escribir el contenido del área de texto en el archivo.

```
private void jMenuItem4ActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setDialogTitle( dialogTitle: "Guardar");
    fileChooser.setFileFilter(new FileNameExtensionFilter( description: "Archivos DF (*.df)",  extensions: "df"));
    int PestanaSeleccionada = jTabbedPane1.getSelectedIndex();
    if (PestanaSeleccionada == -1) {
        JOptionPane.showMessageDialog( parentComponent: this,  message: "No Hay Pestañas Abiertas.",  title: "Guardar",  messageType: JOptionPane.ERROR_MESSAGE);
        return;
    }

    int returnValue = fileChooser.showSaveDialog( parent: null);
    if (returnValue == JFileChooser.APPROVE_OPTION) {
        File archivo = fileChooser.getSelectedFile();
        JTextArea textArea = (JTextArea) ((JScrollPane) jTabbedPane1.getComponentAt( index: PestanaSeleccionada)).getViewport().getView();
        try (BufferedWriter writer = new BufferedWriter(new FileWriter( file: archivo))) {
            writer.write( str: textArea.getText());
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Archivo Guardado Correctamente",  title: "Guardar",  messageType: JOptionPane.INFORMATION_MESSAGE);
        } catch (IOException e) {
            JOptionPane.showMessageDialog( parentComponent: null, "Error: " + e,  title: "Error Guardar",  messageType: JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

Eliminar Pestaña

1. Obtención de la Pestaña Seleccionada:

Se obtiene el índice de la pestaña actualmente seleccionada en el componente jTabbedPane1.

2. Verificación de Pestaña Seleccionada:

Si hay una pestaña seleccionada (selectedIndex != -1), se procede a mostrar un cuadro de diálogo de confirmación para asegurar que el usuario desea eliminar la pestaña.

3. Confirmación del Usuario:

Se muestra un cuadro de diálogo de confirmación (JOptionPane.showConfirmDialog) que pregunta al usuario si está seguro de eliminar la pestaña. El mensaje incluye el título de la pestaña seleccionada (jTabbedPane1.getTitleAt(selectedIndex)).

4. Eliminación de la Pestaña:

Si el usuario confirma la eliminación (confirm == JOptionPane.YES_OPTION), se procede a eliminar la pestaña seleccionada del componente jTabbedPane1.

5. Información al Usuario:

Se muestra un mensaje informativo indicando que la pestaña ha sido eliminada correctamente.

6. Manejo de Caso sin Pestañas:

Si no hay ninguna pestaña abierta (selectedIndex == -1), se muestra un mensaje de error indicando que no hay pestañas abiertas.

```
private void jMenuItem9ActionPerformed(java.awt.event.ActionEvent evt) {
    int selectedIndex = jTabbedPane1.getSelectedIndex();
    if (selectedIndex != -1) {
        int confirm = JOptionPane.showConfirmDialog( parentComponent: this, "¿Estás Seguro De Que Desea Eliminar La Pestaña? " + jTabbedPane1.getTitleAt( index: selectedIndex) ,
        if (confirm == JOptionPane.YES_OPTION) {
            jTabbedPane1.remove( index: selectedIndex);
            JOptionPane.showMessageDialog( parentComponent: this,  message: "Pestaña Eliminada Correctamente",  title: "Eliminar Pestaña",  messageType: JOptionPane.INFORMATION_MESSAGE);
        }
    } else {
        JOptionPane.showMessageDialog( parentComponent: this,  message: "No Hay Pestañas Abiertas.",  title: "Eliminar Pestaña",  messageType: JOptionPane.ERROR_MESSAGE);
    }
}
```

Reportes Tokens

1. Verificación de Lista de Tokens Vacía:

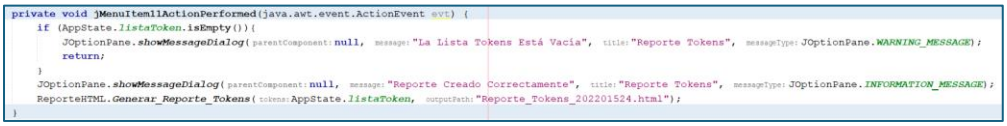
Se verifica si la lista de tokens (AppState.listaToken) está vacía utilizando AppState.listaToken.isEmpty().

2. Manejo de Lista de Tokens Vacía:

Si la lista de tokens está vacía, se muestra un mensaje de advertencia indicando que la lista está vacía y se interrumpe el proceso.

3. Creación del Reporte de Tokens:

Se invoca el método ReporteHTML.Generar_Reporte_Tokens para generar un reporte HTML de los tokens, pasando la lista de tokens y un nombre de archivo como parámetros.



Reporte Tabla De Símbolos

1. Verificación de Tabla de Símbolos Vacía:

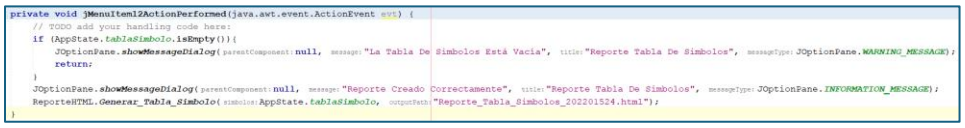
Se verifica si la tabla de símbolos (AppState.tablaSimbolo) está vacía utilizando AppState.tablaSimbolo.isEmpty().

2. Manejo de Tabla de Símbolos Vacía:

Si la tabla de símbolos está vacía, se muestra un mensaje de advertencia indicando que la tabla está vacía y se interrumpe el proceso.

3. Creación del Reporte de Tabla de Símbolos:

Se invoca el método ReporteHTML.Generar_Tabla_Simbolo para generar un reporte HTML de la tabla de símbolos, pasando la tabla de símbolos y un nombre de archivo como parámetros.

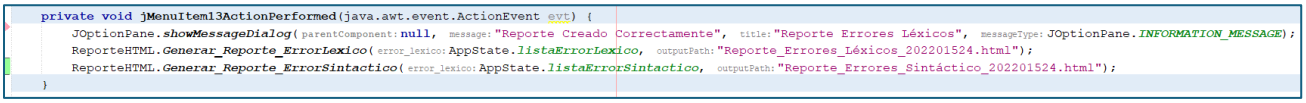


Reporte Errores Léxicos Y Sintácticos

1. Creación de Reportes de Errores:

Se invoca el método ReporteHTML.Generar_Reporte_ErrorLexico para generar un reporte HTML de errores léxicos, pasando la lista de errores léxicos y un nombre de archivo como parámetros.

Se invoca el método ReporteHTML.Generar_Reporte_ErrorSintactico para generar un reporte HTML de errores sintácticos, pasando la lista de errores sintácticos y un nombre de archivo como parámetros.



Ejecutar Análisis

1. Obtención de la Pestaña Seleccionada:

Se obtiene el índice de la pestaña actualmente seleccionada en el componente jTabbedPane1.

2. Verificación de Pestaña Seleccionada:

Si no hay ninguna pestaña abierta (PestanaSeleccionada == -1), se muestra un mensaje de error indicando que no hay pestañas abiertas y se interrumpe el proceso.

3. Limpieza de Estructuras de Datos y Componentes Gráficos:

Se realizan operaciones de limpieza, eliminando el contenido de varias estructuras de datos (AppState.listaToken, AppState.listaErrorLexico, AppState.listaErrorSintactico, AppState.tablaSimbolo) y componentes gráficos (jTextArea1, jTabbedPane2).

4. Obtención del Contenido del Área de Texto:

Se obtiene el contenido del área de texto asociada a la pestaña seleccionada.

5. Ejecución del Análisis Léxico y Sintáctico:

- Se crea un analizador léxico (Lexico) y uno sintáctico (Parser) a partir del contenido del área de texto.
- Se realiza el análisis sintáctico, obteniendo la raíz del árbol sintáctico (ArbolSintactico).
- Se ejecuta el intérprete a partir de la raíz del árbol sintáctico, actualizando las estructuras de datos y componentes gráficos.

6. Impresión de Información en la Consola:

- Se imprime información en la consola relacionada con la tabla de símbolos, tokens y errores sintácticos.

Este método se encarga de realizar el análisis léxico y sintáctico del contenido de la pestaña seleccionada, ejecutar el intérprete, actualizar estructuras de datos y componentes gráficos, imprimir información en la consola y manejar posibles errores.

```
private void jMenu4MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int PestanaSeleccionada = jTabbedPane1.getSelectedIndex();
    if (PestanaSeleccionada == -1) {
        JOptionPane.showMessageDialog( parentComponent: this,  message: "No Hay Pestañas Abiertas.",  title: "Ejecutar Análisis",  messageType: JOptionPane.ERROR_MESSAGE);
        return;
    }
    try {
        AppState.listaToken.clear();
        AppState.listaErrorLexico.clear();
        AppState.listaErrorSintactico.clear();
        AppState.tablaSimbolo.clear();
        jTextArea1.setText( t: "");
        jTabbedPane2.removeAll();
        int selectedIndex =jTabbedPane1.getSelectedIndex();
        JTextArea textArea = (JTextArea) ((JScrollPane) jTabbedPane1.getComponentAt( index: selectedIndex)).getViewport().getView();
        String cadena_entrada = textArea.getText();
        Lexico lexico = new Lexico(new BufferedReader(new StringReader( s: cadena_entrada)));
        Parser sintactico = new Parser( s: lexico);
        ArbolSintactico Raiz = (ArbolSintactico)sintactico.parse().value;
        Raiz.EjecutarInterprete(Raiz,  Consola: jTextArea1,  TablaS: AppState.tablaSimbolo,  tabbedPane: jTabbedPane2);
        System.out.println( x: "-----------------------------------------------");
        AppState.listaToken.addAll( c: lexico.tokens);
        AppState.listaErrorLexico.addAll( c: lexico.lexicalErrors);
        AppState.listaErrorSintactico.addAll( c: sintactico.syntaxErrors);
        for (SimboloNodo Simbolo : AppState.tablaSimbolo) {
            System.out.println("Nombre: " + Simbolo.getNombre() +" -- Tipo: "+ Simbolo.getTipo()+" -- Valor: "+Simbolo.getValor()+" -- Rol: "+Simbolo.getRol());
        }
        System.out.println( x: "-----------------------------------------------");
        for (Token token : AppState.listaToken) {
            System.out.println("Lexema: " + token.getLexema() +" -- Token: "+ token.getTipo());
        }
        System.out.println( x: "-----------------------------------------------");
        for (ErrorSintactico error : AppState.listaErrorSintactico) {
            System.out.println("Tipo: " + error.getTipo() +" -- Token: "+ error.getToken()+" -- Linea: "+ error.getLinea()+" -- Columna: "+ error.getColumna());
        }
```

ÁRBOL SINTÁCTICO

La clase ArbolSintactico es parte de un sistema de análisis sintáctico para un lenguaje específico y ejecución de operaciones en un contexto semántico. A continuación, se presenta una explicación técnica de las operaciones realizadas en esta clase:

Atributos:

- Elemento: Representa el elemento actual en el árbol sintáctico.
- Result: Almacena el resultado de la evaluación del árbol sintáctico.
- Hijos: Lista enlazada que contiene los nodos hijos del árbol sintáctico.

Constructor:

- Se inicializa la clase con un elemento y se crea una lista vacía de hijos.

Método AgregarHijo:

- Añade un nodo hijo al árbol sintáctico.

Método BuscarVariable:

- Busca una variable en una tabla de símbolos y retorna su valor.
- Si no encuentra la variable, retorna una advertencia de error sematico.

Método realizarOperacion:

- Realiza operaciones matemáticas (SUM, RES, MUL, DIV, MOD) a partir de dos números representados como cadenas.
- Maneja errores como la división entre cero o la conversión fallida.

Métodos de Cálculos Estadísticos:

- Realizan cálculos estadísticos a partir de cadenas de números: calcularMedia, calcularMediana, calcularModa, calcularVarianza, encontrarValorMaximo, encontrarValorMinimo.

Método imprimirVariable:

- Formatea y devuelve una cadena resultante de la impresión de variables.

Método imprimirArreglo:

- Formatea y devuelve una cadena resultante de la impresión de un arreglo.

Método EjecutarInterprete:

- Método principal para la ejecución del intérprete.
- Recorre el árbol sintáctico, ejecutando acciones específicas según la estructura del árbol y actualizando la consola, la tabla de símbolos y el panel de pestañas.

Notas Adicionales:

- Se utilizan métodos estáticos para las operaciones matemáticas y estadísticas.
- Se manejan excepciones para garantizar una ejecución robusta ante posibles errores.
- Se realiza un uso de condiciones lógicas y bucles para la ejecución condicional de diferentes operaciones según la estructura del árbol sintáctico.

La clase ArbolSintactico se encarga de interpretar y ejecutar instrucciones basadas en la estructura del árbol sintáctico y proporciona funcionalidades para realizar operaciones matemáticas, estadísticas e impresiones.

```
package Instrucciones;
import java.util.LinkedList;
import javax.swing.*;
import java.util.*;
import Reportes.*;
/**
 *
 * @author manuel
 */
public class ArbolSintactico {
    //-----------------------------------------------------------------------------------------------------------------------
    private String Elemento;
    public String Result;
    private LinkedList<ArbolSintactico> Hijos;

    public ArbolSintactico(String Elemento) {
        this.Elemento = Elemento;
        this.Hijos = new LinkedList();
    }
    public void AgregarHijo(ArbolSintactico Hijo){
        this.Hijos.add( e:Hijo);
    }

    public String BuscarVariable(LinkedList<SimboloNodo> TablaS, String Variable){
        for (SimboloNodo Simbolo : TablaS) {
            if(Simbolo.getNombre().equals( anObject:Variable)){
                return Simbolo.getValor();
            }
        }
        return "ERROR SEMANTICO";
    }
```

```
//-----------------------------------------------------------------------------------------------------------------------
public void EjecutarInterprete(ArbolSintactico Raiz, JTextArea Consola, LinkedList<SimboloNodo> TablaS, JTabbedPane tabbedPane){
    for (ArbolSintactico Hijo: Raiz.Hijos){
        EjecutarInterprete( Raiz:Hijo, Consola, TablaS, tabbedPane);
    }
    if(Raiz.getElemento()=="ASIGNACION VARIABLE"){
        SimboloNodo SimboloNuevo = new SimboloNodo(
                nombre:Raiz.getHijos().get( index:5).getElemento(),
                tipo:Raiz.getHijos().get( index:2).getElemento(),
                rol:"Variable",
                valor:Raiz.getHijos().get( index:8).Result,
                linea:Raiz.getHijos().get( index:11).getElemento(),
                columna:Raiz.getHijos().get( index:12).getElemento()
        );
        TablaS.add( e:SimboloNuevo);

    }else if(Raiz.getElemento()=="ASIGNACION ARREGLO"){
        SimboloNodo SimboloNuevo = new SimboloNodo(
                nombre:Raiz.getHijos().get( index:5).getElemento(),
                tipo:Raiz.getHijos().get( index:2).getElemento(),
                rol:"Array",
                valor:Raiz.getHijos().get( index:8).Result,
                linea:Raiz.getHijos().get( index:11).getElemento(),
                columna:Raiz.getHijos().get( index:12).getElemento()
        );
        TablaS.add( e:SimboloNuevo);
    }
    else if(Raiz.getElemento()=="ASIGNACION IMPRIMIR" && Raiz.getHijos().size()==8){
        String Variable = imprimirVariable( cadena:Raiz.getHijos().get( index:5).Result);
        Consola.append(Variable+"\n");
    }
    else if(Raiz.getElemento()=="ASIGNACION IMPRIMIR" & Raiz.getHijos().size()==11){
        String Arreglo = imprimirArreglo( titulo:Raiz.getHijos().get( index:5).Result,  lista:Raiz.getHijos().get( index:8).Result);
        Consola.append(Arreglo+"\n");
    }
```

REPORTE HTML

Método Generar_Reporte_Tokens:

- Este método toma una lista enlazada de objetos Token y una ruta de salida como parámetros.
- Crea un directorio llamado "Reportes" si no existe.
- Abre un archivo HTML en la ruta de salida y escribe información sobre los tokens en una tabla dentro del archivo.
- Usa estilos CSS en línea para dar formato al HTML.

Método Generar_Reporte_ErrorLexico:

- Similar al método anterior, pero toma una lista enlazada de objetos ErrorLexico.
- Genera un informe HTML con información detallada sobre los errores léxicos.

Método Generar_Reporte_ErrorSintactico:

- Similar a los métodos anteriores, pero toma una lista enlazada de objetos ErrorSintactico.
- Genera un informe HTML con información detallada sobre los errores sintácticos.

Método Generar_Tabla_Simbolo:

- Similar a los métodos anteriores, pero toma una lista enlazada de objetos SimboloNodo.
- Genera un informe HTML con información detallada sobre los símbolos del programa.
- Estilos CSS en línea:
- Define estilos CSS en línea para dar formato a las tablas y al texto en los informes.

```
package Reportes;
import Instrucciones.SimboloNodo;
import java.io.*;
import java.io.IOException;
import java.util.LinkedList;
import dataforge.Token;
import dataforge.ErrorLexico;
import dataforge.ErrorSintactico;
import java.nio.file.Files;
import java.nio.file.Path;
import javax.swing.JOptionPane;
/**
 *
 *
 * @author manuel
 */
public class ReporteHTML {

    public static void Generar_Reporte_Tokens(LinkedList<Token> tokens, String outputPath) {
        try {
            Path carpetaReporte = Path.of( first: "Reportes");
            if (!Files.exists( path: carpetaReporte)) {
                Files.createDirectory( dir: carpetaReporte);
            }
            Path rutaArchivo = Path.of( first: "Reportes",  more: outputPath);
            try (BufferedWriter codigoHTML = new BufferedWriter(new FileWriter( fileName: rutaArchivo.toString()))) {
                codigoHTML.write( str: "<!DOCTYPE html>\n<html>\n<head>\n");
                codigoHTML.write( str: "<style>\n");
                codigoHTML.write("body {\n"
                        + "    font-family: 'Helvetica', sans-serif;\n"
                        + "    text-align: center;\n"
                        + "}\n");
                codigoHTML.write("table {\n"
                        + "    font-family: 'Helvetica', sans-serif;\n"
                        + "    border-collapse: collapse;\n"
                        + "    width: 70%;\n"
```

GRAFICA BARRAS

Método crearGrafica:

- Toma una cadena de entrada y un objeto JTabbedPane como parámetros.
- Divide la cadena en partes utilizando el carácter "|".
- Extrae información como el título de la gráfica, las etiquetas del eje x e y, y los datos.
- Crea un conjunto de datos (CategoryDataset) llamando al método createDataset y luego utiliza estos datos para crear un objeto JFreeChart representando una gráfica de barras.
- Crea un ChartPanel a partir del objeto JFreeChart.
- Busca un panel existente con el mismo título en el JTabbedPane. Si existe, agrega el ChartPanel al panel existente; de lo contrario, crea un nuevo panel y agrega la gráfica a una nueva pestaña en el JTabbedPane.

Método createDataset:

- Toma dos arrays de cadenas, uno para las etiquetas del eje x (ejex) y otro para los datos del eje y (ejey).
- Utiliza un DefaultCategoryDataset para crear un conjunto de datos categorizado.
- Itera sobre las etiquetas del eje x y agrega los valores correspondientes del eje y al conjunto de datos.

Método obtenerPanelExistente:

- Toma un objeto JTabbedPane y un título como parámetros.
- Itera sobre las pestañas en el JTabbedPane y busca un panel con el mismo título.
- Devuelve el panel existente si lo encuentra; de lo contrario, devuelve null.

```
        CategoryDataset dataset = createDataset(ejex, ejey);
        JFreeChart barChart = ChartFactory.createBarChart(
                title: titulo,
                categoryAxisLabel: tituloX,
                valueAxisLabel: tituloY,
                dataset,
                orientation: PlotOrientation.VERTICAL,
                legend: true,
                tooltips: true,
                urls: false
        );
        ChartPanel chartPanel = new ChartPanel( chart: barChart);
        chartPanel.setPreferredSize(new java.awt.Dimension( width: tabbedPane.getWidth(),  height: tabbedPane.getHeight()));
        JPanel panelExistente = obtenerPanelExistente(tabbedPane, titulo);
        if (panelExistente != null) {
            panelExistente.add( comp: chartPanel);
        } else {
            JPanel nuevoPanel = new JPanel();
            nuevoPanel.add( comp: chartPanel);
            tabbedPane.addTab( title: titulo,  component: nuevoPanel);
        }
        return "Gráfica De Barras '" + titulo + "' Creada Correctamente.";
    }
    private static CategoryDataset createDataset(String[] ejex, String[] ejey) {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        for (int i = 0; i < ejex.length; i++) {
            dataset.addValue( value: Double.parseDouble(ejey[i]),  rowKey: "Datos", ejex[i]);
        }
        return dataset;
    }
```

GRAFICA CIRCULAR

Método crearGraficaCircular:

- Toma una cadena de entrada y un objeto JTabbedPane como parámetros.
- Divide la cadena en partes utilizando el carácter "|".
- Extrae información como el título de la gráfica, las etiquetas y los valores.
- Crea un conjunto de datos (PieDataset) llamando al método createDataset y luego utiliza estos datos para crear un objeto JFreeChart representando una gráfica circular.
- Crea un ChartPanel a partir del objeto JFreeChart.
- Busca un panel existente con el mismo título en el JTabbedPane. Si existe, agrega el ChartPanel al panel existente; de lo contrario, crea un nuevo panel y agrega la gráfica a una nueva pestaña en el JTabbedPane.

Método createDataset:

- Toma dos arrays de datos, uno para las etiquetas (labels) y otro para los valores (values).
- Utiliza un DefaultPieDataset para crear un conjunto de datos de gráfica circular.
- Itera sobre las etiquetas y agrega los valores correspondientes al conjunto de datos.

Método obtenerPanelExistente:

- Toma un objeto JTabbedPane y un título como parámetros.
- Itera sobre las pestañas en el JTabbedPane y busca un panel con el mismo título.
- Devuelve el panel existente si lo encuentra; de lo contrario, devuelve null.

```
    PieDataset dataset = createDataset(labels, values);
    JFreeChart pieChart = ChartFactory.createPieChart(
            title: titulo,
            dataset,
            legend: true,
            tooltips: true,
            urls: false
    );

    ChartPanel chartPanel = new ChartPanel( chart: pieChart);
    chartPanel.setPreferredSize(new java.awt.Dimension( width: tabbedPane.getWidth(),  height: tabbedPane.getHeight()));

    JPanel panelExistente = obtenerPanelExistente(tabbedPane, titulo);
    if (panelExistente != null) {
        panelExistente.add( comp: chartPanel);
    } else {
        JPanel nuevoPanel = new JPanel();
        nuevoPanel.add( comp: chartPanel);
        tabbedPane.addTab( title: titulo,  component: nuevoPanel);
    }
    return "Gráfica Circular '" + titulo + "' Creada Correctamente.";
}
private static PieDataset createDataset(String[] labels, double[] values) {
    DefaultPieDataset dataset = new DefaultPieDataset();
    for (int i = 0; i < labels.length; i++) {
        dataset.setValue(labels[i], values[i]);
    }
    return dataset;
}
private static JPanel obtenerPanelExistente(JTabbedPane tabbedPane, String titulo) {
    for (int i = 0; i < tabbedPane.getTabCount(); i++) {
        if (tabbedPane.getTitleAt( index: i).equals( anObject: titulo)) {
            return (JPanel) tabbedPane.getComponentAt( index: i);
        }
    }
    return null;
}
```

GRAFICA HISTOGRAMA

Método crearHistograma:

- Toma una cadena de entrada y un objeto JTabbedPane como parámetros.
- Divide la cadena en partes utilizando el carácter "|".
- Extrae información como el título del histograma y los valores.
- Crea un conjunto de datos (DefaultCategoryDataset) llamando al método createDataset y luego utiliza estos datos para crear un objeto JFreeChart representando un histograma.
- Crea un ChartPanel a partir del objeto JFreeChart.
- Busca un panel existente con el mismo título en el JTabbedPane. Si existe, agrega el ChartPanel al panel existente; de lo contrario, crea un nuevo panel y agrega la gráfica a una nueva pestaña en el JTabbedPane.
- Genera información sobre el análisis del arreglo y la imprime en la consola.

Método createDataset:

- Toma un array de valores como parámetro.
- Utiliza un DefaultCategoryDataset para crear un conjunto de datos para el histograma.
- Calcula las frecuencias absolutas y relativas de cada valor y las agrega al conjunto de datos.

Método obtenerPanelExistente:

- Toma un objeto JTabbedPane y un título como parámetros.
- Itera sobre las pestañas en el JTabbedPane y busca un panel con el mismo título.
- Devuelve el panel existente si lo encuentra; de lo contrario, devuelve null.

Método obtenerInformacion:

- Toma un array de valores como parámetro.
- Calcula las frecuencias absolutas y relativas de cada valor.
- Construye y devuelve una cadena de información detallada sobre el análisis del arreglo, incluyendo valores, frecuencias absolutas, frecuencias acumuladas y frecuencias relativas.

```
ChartPanel chartPanel = new ChartPanel( chart: barChart);
chartPanel.setPreferredSize(new java.awt.Dimension( width: tabbedPane.getWidth(),  height: tabbedPane.getHeight()));

JPanel panelExistente = obtenerPanelExistente(tabbedPane, titulo);
if (panelExistente != null) {
    panelExistente.add( comp: chartPanel);
} else {
    JPanel nuevoPanel = new JPanel();
    nuevoPanel.add( comp: chartPanel);
    tabbedPane.addTab( title: titulo,  component: nuevoPanel);
}
String informacion = obtenerInformacion(values);
System.out.println( x: informacion);
return "Gráfica Histograma '" + titulo + "' Creada Correctamente.\n"+informacion;
```

GRAFICA LINEAL

Método crearGraficaLineas:

- Toma una cadena de entrada y un objeto JTabbedPane como parámetros.
- Divide la cadena en partes utilizando el carácter "|".
- Extrae información como el título de la gráfica, los ejes X e Y, y las etiquetas de los datos.
- Crea un conjunto de datos (DefaultCategoryDataset) llamando al método createDataset y luego utiliza estos datos para crear un objeto JFreeChart representando una gráfica de líneas.
- Crea un ChartPanel a partir del objeto JFreeChart.
- Busca un panel existente con el mismo título en el JTabbedPane. Si existe, agrega el ChartPanel al panel existente; de lo contrario, crea un nuevo panel y agrega la gráfica a una nueva pestaña en el JTabbedPane.

Método createDataset:

- Toma arrays de ejes X y Y como parámetros.
- Utiliza un DefaultCategoryDataset para crear un conjunto de datos para la gráfica de líneas.
- Agrega los valores de los ejes X e Y al conjunto de datos.

Método obtenerPanelExistente:

- Toma un objeto JTabbedPane y un título como parámetros.
- Itera sobre las pestañas en el JTabbedPane y busca un panel con el mismo título.
- Devuelve el panel existente si lo encuentra; de lo contrario, devuelve null.

```
    CategoryDataset dataset = createDataset(ejex, ejey);
    JFreeChart lineChart = ChartFactory.createLineChart(
            title: titulo,
            categoryAxisLabel: tituloX,
            valueAxisLabel: tituloY,
            dataset,
            orientation: PlotOrientation.VERTICAL,
            legend: true,
            tooltips: true,
            urls: false
    );
    ChartPanel chartPanel = new ChartPanel( chart: lineChart);
    chartPanel.setPreferredSize(new java.awt.Dimension( width: tabbedPane.getWidth(),  height: tabbedPane.getHeight()));
    JPanel panelExistente = obtenerPanelExistente(tabbedPane, titulo);
    if (panelExistente != null) {
        panelExistente.add( comp: chartPanel);
    } else {
        JPanel nuevoPanel = new JPanel();
        nuevoPanel.add( comp: chartPanel);
        tabbedPane.addTab( title: titulo,  component: nuevoPanel);
    }
    return "Gráfica Lineal '" + titulo + "' Creada Correctamente.";
}

private static CategoryDataset createDataset(String[] ejex, String[] ejey) {
    DefaultCategoryDataset dataset = new DefaultCategoryDataset();
    for (int i = 0; i < ejex.length; i++) {
        dataset.addValue( value: Double.parseDouble(ejey[i]),  rowKey: "Datos", ejex[i]);
    }
    return dataset;
}
```

GENERADOR LÉXICO Y SINTÁCTICO

Esta clase Java, llamada Generador, se utiliza para generar los analizadores léxico y sintáctico a partir de archivos JFlex y Cup respectivamente. Aquí tienes una explicación técnica del código:

Método main:

1. Este método es la entrada principal del programa.
2. Utiliza un bloque try-catch para manejar excepciones que puedan ocurrir durante la generación de analizadores léxico y sintáctico.
3. Define una cadena ruta que representa la ubicación de los archivos de definición (JFlex y Cup).
4. Define un array de cadenas opJFlex que contiene los argumentos para la generación del analizador léxico con JFlex. Estos argumentos incluyen la ubicación del archivo de definición JFlex (lexico.jflex) y la ubicación de salida (-d).
5. Llama al generador de JFlex (jflex.Main.generate(opJFlex)).
6. Define un array de cadenas opCup que contiene los argumentos para la generación del analizador sintáctico con Cup. Estos argumentos incluyen la ubicación del archivo de definición Cup (sintactico.cup), la ubicación de salida (-destdir), y el nombre del parser generado (-parser).
7. Llama al generador de Cup (java_cup.Main.main(opCup)).
8. Captura y maneja cualquier excepción que pueda ocurrir durante el proceso, imprimiendo información sobre la excepción.

En resumen, esta clase Java proporciona un programa principal (main) que utiliza JFlex y Cup para generar los analizadores léxico y sintáctico necesarios para procesar un lenguaje específico. La ubicación de los archivos de definición y los argumentos de generación se especifican en el código.

```java
package AnalizadoresLexicoSintactico;
/**
 *
 * @author manuel
 */
public class Generador {
    public static void main(String[] args) {
        try {
            String ruta = "./src/AnalizadoresLexicoSintactico/";
            String[] opJFlex = {ruta+"lexico.jflex","-d",ruta};
            jflex.Main.generate( argv: opJFlex);
            String[] opCup = {"-destdir",ruta,"-parser","Parser",ruta+"sintactico.cup"};
            java_cup.Main.main( argv: opCup);
        } catch (Exception e) {
            System.out.println( x: e);
        }
    }
}
```

ANALIZADOR LÉXICO

Encabezado:

- Importa las clases necesarias.
- Define un paquete llamado AnalizadoresLexicoSintactico.

Sección de opciones de JFlex:

- %class Lexico: Indica el nombre de la clase generada por JFlex.
- %public: Hace que la clase sea pública.
- %line y %column: Habilitan el seguimiento de líneas y columnas.
- %cup: Indica que se está integrando con CUP (parser generator).
- %unicode: Habilita el uso de Unicode.
- %ignorecase: Ignora las diferencias entre mayúsculas y minúsculas.

Sección de código en Java:

- Se utilizan bloques %{...%} para incluir código Java dentro del archivo de especificación.
- Se crean variables públicas lexicalErrors y tokens para almacenar errores léxicos y tokens respectivamente.
- Se inicializan las variables yyline y yycolumn.

Definición de expresiones regulares:

Se definen patrones de expresiones regulares utilizando notación de JFlex. Estos patrones se utilizan para identificar lexemas en el código fuente.

Reglas para reconocer tokens:

- Se definen reglas que coinciden con ciertos patrones y generan tokens correspondientes.
- Cada regla está precedida por un patrón y seguida por un bloque de código Java que realiza acciones asociadas a la identificación del token.
- Los tokens y sus tipos se definen usando tokens.add(new Token(...)).

Reglas de acción para otros elementos:

Se definen reglas de acción para elementos como espacios en blanco, comentarios, y otros elementos que deben ser ignorados por el analizador léxico.

Manejo de errores léxicos:

Se utilizan reglas para manejar errores léxicos. Si no se puede asociar un lexema a ningún patrón, se registra un error léxico.

Reglas de producción para ciertos lexemas:

Se definen reglas de producción utilizando las expresiones regulares definidas anteriormente.

Reglas de acción para otros elementos:

Se definen reglas de acción para elementos como espacios en blanco, comentarios, y otros elementos que deben ser ignorados por el analizador léxico.

Reglas de acción para otros elementos:

Se definen reglas de acción para elementos como espacios en blanco, comentarios, y otros elementos que deben ser ignorados por el analizador léxico.

```
package AnalizadoresLexicoSintactico;
import java.util.LinkedList;
import dataforge.Token;
import java_cup.runtime.Symbol;
import dataforge.ErrorLexico;

//------------------------------------------------------------------------------
%%
%class Lexico
%public
%line
%char
%cup
%unicode
%ignorecase
%column

%{
public LinkedList<ErrorLexico> lexicalErrors;
public LinkedList<Token> tokens;
%}

%init{
lexicalErrors = new LinkedList<>();
tokens = new LinkedList<>();
yyline = 1;
yycolumn = 1;
%init}

//------------------------------------------------------------------------------
EN_BLANCO = [ \r\t]+
COMENTARIO_LINEA = ("!".*\r\n)|("!".*\n)|("!".*\r)
SIMBOLO_EXCLAMAC = [^!]
COMENTARIO_MULTI = "<!"{SIMBOLO_EXCLAMAC}* "!>"
```

Analizador Sintáctico

Declaraciones Iniciales:

El bloque parser code incluye funciones y variables adicionales que afectan el comportamiento del analizador sintáctico. En este caso, se define una lista enlazada llamada syntaxErrors para almacenar errores sintácticos.

Se definen dos métodos: syntax_error y unrecovered_syntax_error. El primero se utiliza para registrar errores sintácticos en la lista mencionada, mientras que el segundo se llama cuando el analizador sintáctico no puede recuperarse de un error.

Terminales:

Se definen varios terminales utilizando la palabra clave terminal. Estos representan unidades léxicas o tokens que el analizador sintáctico reconocerá en el código fuente, como palabras clave y símbolos.}

Reglas de Producción:

Se definen las reglas de producción que describen cómo se deben construir las diferentes partes del lenguaje. Estas reglas especifican cómo se forman las declaraciones, asignaciones y otros elementos sintácticos.

Especificación del Inicio:

La producción start with inicio; indica el punto de inicio del análisis sintáctico, donde la ejecución del analizador comienza.

Reglas de Producción Detalladas:

Se definen reglas de producción más específicas para declaraciones, asignaciones, impresiones, asignaciones de gráficos, etc. Cada regla tiene una acción asociada escrita en bloques {} que crea un árbol sintáctico (ArbolSintactico) para representar la estructura sintáctica de la entrada.

Manejo de Errores:

Se manejan errores sintácticos a través de las acciones definidas en el bloque parser code. Cuando se encuentra un error, se registra en la lista syntaxErrors.

```
package AnalizadoresLexicoSintactico;
import java_cup.runtime.*;
import Instrucciones.ArbolSintactico;
import java.util.LinkedList;
import dataforge.ErrorSintactico;

parser code
{:
    public LinkedList<ErrorSintactico> syntaxErrors = new LinkedList<>();

    public void syntax_error(Symbol s){
        if (s.value != null){
            syntaxErrors.add(new ErrorSintactico("Sintactico", s.value.toString(), s.left, s.right));
        }
    }
    public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception{
        System.out.println("Error Sintactico No Recuperado: "+s.value+" , linea: "+s.left+" columna: "+s.right);
    }
:}
```