UNIVERSIDAD DE SAN CARLOS DE GUATEMALA FACULTAD DE INGENIERÍA, ESCUELA DE CIENCIAS Y SISTEMAS

OLC1

Proyecto 1 - Manual Técnico

Diego Josue Guevara Abaj

Fecha: 9 de marzo de 2024

Introducción

Bienvenido al manual de Tecnico del sistema de construcción de analizadores léxicos y sintácticos desarrollado para el curso de Organización de Lenguajes y Compiladores 1 de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala. Este manual está diseñado para proporcionar una guía detallada sobre cómo utilizar esta herramienta para aplicar los conocimientos adquiridos sobre la fase de análisis léxico y sintáctico de un compilador en la construcción de soluciones de software.

Objetivo General

El objetivo principal de este sistema es permitir a los estudiantes aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para la construcción de soluciones de software. Con este sistema, los usuarios podrán generar analizadores léxicos y sintácticos utilizando las herramientas JFLEX y CUP, comprender los conceptos de token, lexema, patrones y expresiones regulares, manejar correctamente los errores léxicos y realizar acciones gramaticales utilizando el lenguaje de programación JAVA.

Objetivos Específicos

1. Aprender a generar analizadores léxicos y sintácticos utilizando las herramientas de JFLEX y CUP.
2. Comprender los conceptos de token, lexema, patrones y expresiones regulares.
3. Realizar correctamente el manejo de errores léxicos.
4. Ser capaz de realizar acciones gramaticales utilizando el lenguaje de programación JAVA.

Descripción General

Este sistema fue desarrollado para cumplir con los requisitos del curso de Organización de Lenguajes y Compiladores 1, y se espera que sea utilizado por los

estudiantes como parte de su aprendizaje en la construcción de analizadores léxicos y sintácticos. El sistema es capaz de realizar operaciones aritméticas y estadísticas, además de generar diversos gráficos a partir de una colección de datos.

Para comenzar a utilizar este sistema, los usuarios deben familiarizarse con el entorno de trabajo proporcionado, que incluye un editor integrado para la creación y edición de archivos de código fuente.

Requisitos del Sistema

Requisitos Mínimos:

- Sistema Operativo: Ubuntu 22 (u otra distribución Linux compatible) o Windows 10/11.
- Memoria RAM: 4 GB (Se recomiendan 8 GB o más para un rendimiento óptimo).
- Procesador: Procesador de doble núcleo a 2.0 GHz o superior.
- Espacio en Disco Duro: Al menos 500 MB de espacio disponible.
- Java Development Kit (JDK) 17 instalado y configurado correctamente en el sistema.

Requisitos Recomendados:

- Sistema Operativo: Ubuntu 22 o Windows 10/11.
- Memoria RAM: 8 GB o más.
- Procesador: Procesador de cuatro núcleos a 2.5 GHz o superior.
- Espacio en Disco Duro: 1 GB de espacio disponible.
- Java Development Kit (JDK) 17 instalado y configurado correctamente en el sistema.

Dado que el tamaño del proyecto es relativamente pequeño, no debería tener un impacto significativo en los requisitos del sistema. La cantidad de RAM y el tipo de procesador serán los principales factores a considerar para un rendimiento óptimo, junto con la disponibilidad de espacio en disco y la instalación adecuada del JDK.

Instalación

La instalación del programa en un entorno Ubuntu 22 (u otra distribución Linux compatible) y en Windows 10/11 será bastante similar, dado que el programa está desarrollado en Java y es compatible con ambos sistemas operativos. Aquí tienes una guía básica para la instalación en ambos sistemas:

En Ubuntu 22 (o distribución Linux compatible):

Instalación del JDK 17:

1. Abre una terminal.
2. Ejecuta el siguiente comando para instalar el JDK 17:

   sudo apt-get install openjdk-17-jdk

· **Descarga del programa:**

· Descarga el archivo del programa desde la fuente proporcionada.

Puedes guardar el archivo en tu carpeta de inicio o en cualquier otro directorio de tu elección.

· **Extracción del archivo (si es necesario):**

· Si el archivo se descargó en formato comprimido (por ejemplo, .zip), deberás extraer su contenido.

Puedes hacerlo haciendo clic derecho sobre el archivo y seleccionando "Extraer aquí" o utilizando el comando unzip en la terminal.

· **Ejecución del programa:**

· Abre una terminal en la ubicación donde se encuentra el archivo ejecutable del programa.

Ejecuta el programa utilizando el comando java -jar nombre_del_archivo.jar.

Por ejemplo: java -jar programa.jar

En Windows 10/11:

Instalación del JDK 17:

1. Descarga el instalador del JDK 17 desde el sitio web oficial de Oracle o AdoptOpenJDK.
2. Ejecuta el instalador y sigue las instrucciones en pantalla para completar la instalación.

Descarga del programa:

1. Descarga el archivo del programa desde la fuente proporcionada.
2. Puedes guardar el archivo en tu carpeta de Descargas o en cualquier otro directorio de tu elección.

Ejecución del programa:

1. Abre el Explorador de Archivos y navega hasta la ubicación donde se encuentra el archivo ejecutable del programa.
2. Haz doble clic en el archivo ejecutable (con extensión .jar) para ejecutar el programa.

Siguiendo estos pasos, podrás instalar y ejecutar el programa en ambos sistemas operativos. Asegúrate de tener instalado el JDK 17 en tu sistema antes de intentar ejecutar el programa.

En Windows 10/11:

Instalación del JDK 17:

1. Descarga el instalador del JDK 17 desde el sitio web oficial de Oracle o AdoptOpenJDK.
2. Ejecuta el instalador y sigue las instrucciones en pantalla para completar la instalación.

Descarga del programa:

1. Descarga el archivo del programa desde la fuente proporcionada.
2. Puedes guardar el archivo en tu carpeta de Descargas o en cualquier otro directorio de tu elección.

Ejecución del programa:

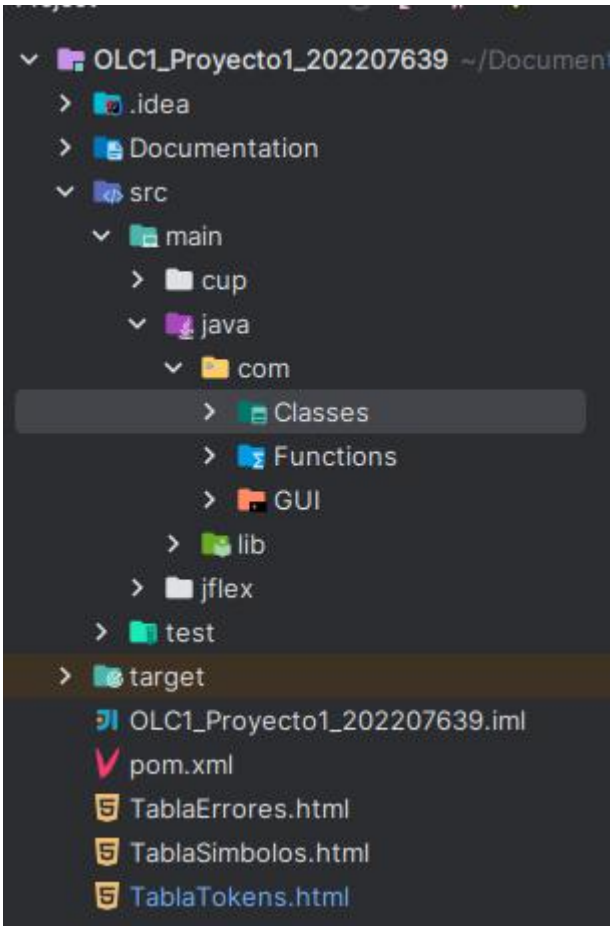
1. Abre el Explorador de Archivos y navega hasta la ubicación donde se encuentra el archivo ejecutable del programa.
2. Haz doble clic en el archivo ejecutable (con extensión .jar) para ejecutar el programa.

Siguiendo estos pasos, podrás instalar y ejecutar el programa en ambos sistemas operativos. Asegúrate de tener instalado el JDK 17 en tu sistema antes de intentar ejecutar el programa.

Arquitectura del código

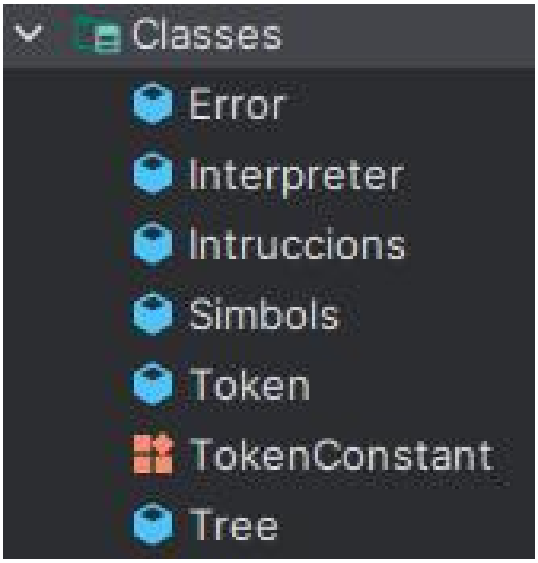
El proyecto se encuentra dividido en los siguientes directorios, En donde se contiene toda la información necesaria para el entendimiento del proyecto.

- Documentación: Directorio en donde se encuentra tando el manual técnico como el manual de usuario, tambien se encuentra el diagrama de clases.
- En src, se encuenetra contenido todo el proyecto principal en su mayoria, conteniendo los archivos .cup, .flex, y los relacionados con clases de java, funciones e interfaces.
- En target: se encuentran los analizadores tanto sintactico como lexico.



Classes:

En el apartado de las clases contamos con las siguientes: Error, Interpreter, Instrucions, Simbols, Token, TokenConstant, Tree



Error:

La clase `Error` es una clase en Java que representa un error. Tiene varios campos y métodos para manejar la información relacionada con un error.

Campos:

- `id`: Un entero que representa el identificador único del error.

- `line`: Un entero que representa la línea en la que ocurrió el error.

- `column`: Un entero que representa la columna en la que ocurrió el error.

- `character`: Una cadena que representa el carácter donde ocurrió el error.

- `ErrorType`: Una cadena que representa el tipo de error.

- `Description`: Una cadena que describe el error.

Métodos:

- `getId()`: Devuelve el identificador del error.

- `setId(int id)`: Establece el identificador del error.

- `getLine()`: Devuelve la línea en la que ocurrió el error.

- `setLine(int line)`: Establece la línea en la que ocurrió el error.

- `getColumn()`: Devuelve la columna en la que ocurrió el error.

- `setColumn(int column)`: Establece la columna en la que ocurrió el error.

- `getCharacter()`: Devuelve el carácter donde ocurrió el error.

- `setCharacter(String character)`: Establece el carácter donde ocurrió el error.

- `getErrorType()`: Devuelve el tipo de error.

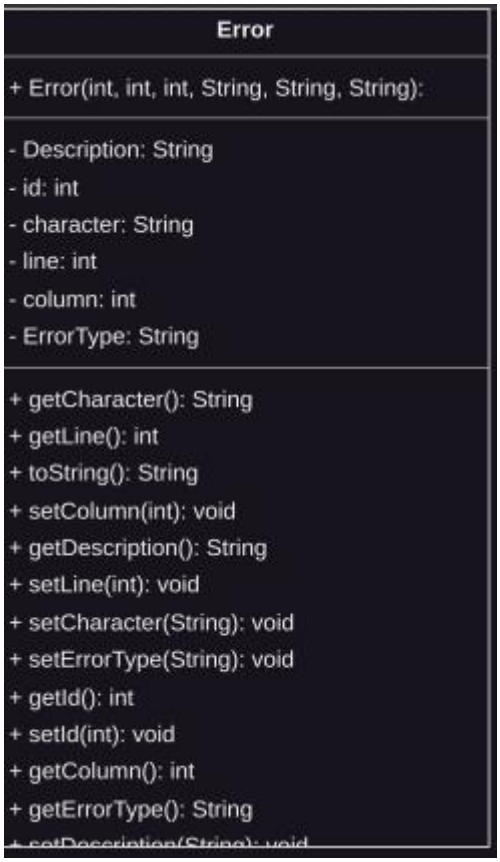
- `setErrorType(String errorType)`: Establece el tipo de error.

- `getDescription()`: Devuelve la descripción del error.

- `setDescription(String description)`: Establece la descripción del error.

- `toString()`: Devuelve una representación en cadena de la clase `Error`.

La clase `Error` se utiliza para encapsular la información relacionada con un error en un solo objeto. Esto facilita el manejo de errores y la depuración.



Tokens:

La clase `Token` es una clase en Java que representa un token. Tiene varios campos y métodos para manejar la información relacionada con un token.

Campos:

- `id`: Un entero que representa el identificador único del token.

- `line`: Un entero que representa la línea en la que se encontró el token.

- `column`: Un entero que representa la columna en la que se encontró el token.

- `lexeme`: Una cadena que representa el lexema del token.

- `regularExpression`: Una cadena que representa la expresión regular del token.

- `tokenType`: Un objeto de tipo `TokenConstant` que representa el tipo de token.

Métodos:

- `getId()`: Devuelve el identificador del token.

- `setId(int id)`: Establece el identificador del token.

- `getLine()`: Devuelve la línea en la que se encontró el token.

- `setLine(int line)`: Establece la línea en la que se encontró el token.

- `getColumn()`: Devuelve la columna en la que se encontró el token.

- `setColumn(int column)`: Establece la columna en la que se encontró el token.

- `getLexeme()`: Devuelve el lexema del token.

- `setLexeme(String lexeme)`: Establece el lexema del token.

- `getRegularExpression()`: Devuelve la expresión regular del token.

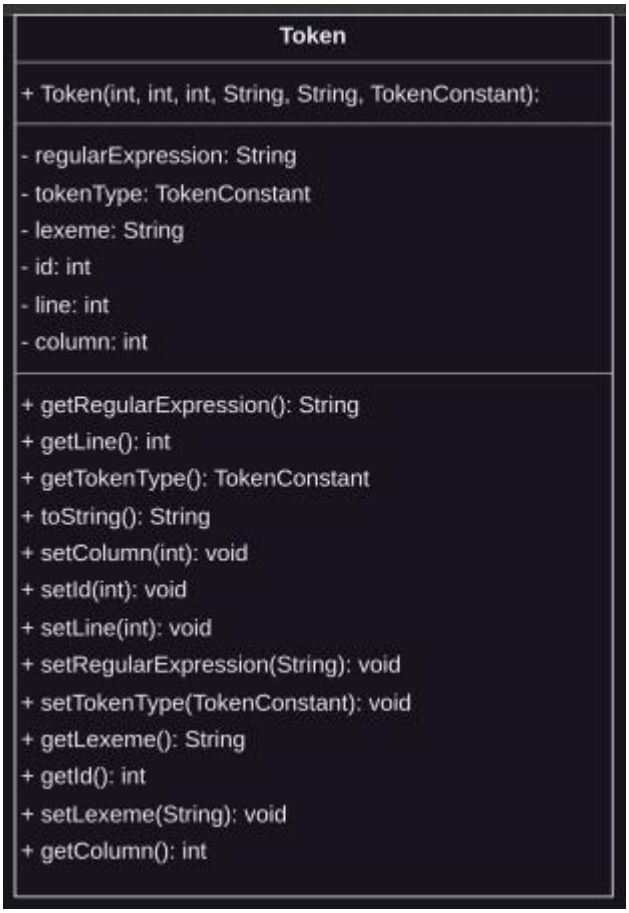
- `setRegularExpression(String regularExpression)`: Establece la expresión regular del token.

- `getTokenType()`: Devuelve el tipo de token.

- `setTokenType(TokenConstant tokenType)`: Establece el tipo de token.

- `toString()`: Devuelve una representación en cadena de la clase `Token`.

La clase `Token` se utiliza para encapsular la información relacionada con un token en un solo objeto. Esto facilita el manejo de tokens y la depuración.



Arbol:

La clase `Tree` es una clase en Java que representa un árbol. Tiene varios campos y métodos para manejar la información relacionada con un árbol.

Campos:

- `fail`: Un booleano que representa si el árbol ha fallado.

- `line`: Un entero que representa la línea en la que se encuentra el árbol.

- `column`: Un entero que representa la columna en la que se encuentra el árbol.

- `lexeme`: Una cadena que representa el lexema del árbol.

- `children`: Una lista de árboles que son hijos de este árbol.

- `Instruccions`: Una lista de instrucciones.

- `Ignore`: Una lista de cadenas que se deben ignorar.

Métodos:

- `Tree(String lexeme, int line, int column)`: Constructor de la clase `Tree`.

- `addChild(Tree child)`: Añade un hijo al árbol.

- `printTree(Tree seed)`: Imprime el árbol.

- `saveTree(Tree seed)`: Guarda el árbol.

- `getInstruccions(Tree seed)`: Devuelve las instrucciones del árbol.

- `printInstruccions()`: Imprime las instrucciones.

La clase `Tree` se utiliza para encapsular la información relacionada con un árbol en un solo objeto. Esto facilita el manejo de árboles y la depuración.

Instruccions:

La clase `Intruccions` es una clase en Java que representa una instrucción. Tiene varios campos y métodos para manejar la información relacionada con una instrucción.

Campos:

- `line`: Un entero que representa la línea en la que se encuentra la instrucción.

- `column`: Un entero que representa la columna en la que se encuentra la instrucción.

- `lexeme`: Una cadena que representa el lexema de la instrucción.

Métodos:

- `Intruccions(int line, int column, String lexeme)`: Constructor de la clase `Intruccions`.

- `getLine()`: Devuelve la línea en la que se encuentra la instrucción.

- `setLine(int line)`: Establece la línea en la que se encuentra la instrucción.

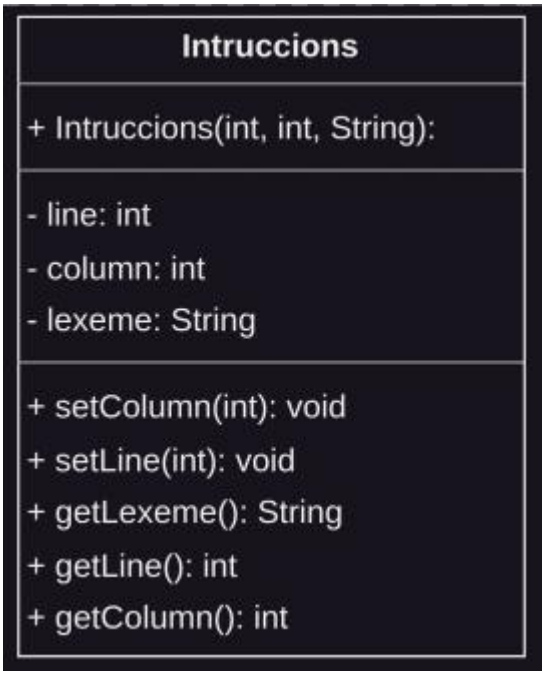
- `getColumn()`: Devuelve la columna en la que se encuentra la instrucción.

- `setColumn(int column)`: Establece la columna en la que se encuentra la instrucción.

- `getLexeme()`: Devuelve el lexema de la instrucción.

- `setLexeme(String lexeme)`: Establece el lexema de la instrucción.

La clase `Intruccions` se utiliza para encapsular la información relacionada con una instrucción en un solo objeto. Esto facilita el manejo de instrucciones y la depuración.



CombinedGraphs:

La clase `CombinedGraphs` es una clase en Java que se utiliza para crear diferentes tipos de gráficos utilizando la biblioteca JFreeChart. Tiene varios métodos para crear gráficos de barras, gráficos de pastel, gráficos de líneas y gráficos de barras de frecuencia.

Campos:

- `chartPanels`: Una lista de `ChartPanel` que almacena los gráficos creados.

Métodos:

- `createBarGraph(String title, ArrayList<String> ejeX, ArrayList<Float> ejeY, String tituloX, String tituloY)`: Crea un gráfico de barras con los datos proporcionados y devuelve un `ChartPanel`.

- `createPieGraph(String title, ArrayList<String> labels, ArrayList<Float> values)`: Crea un gráfico de pastel con los datos proporcionados y devuelve un `ChartPanel`.

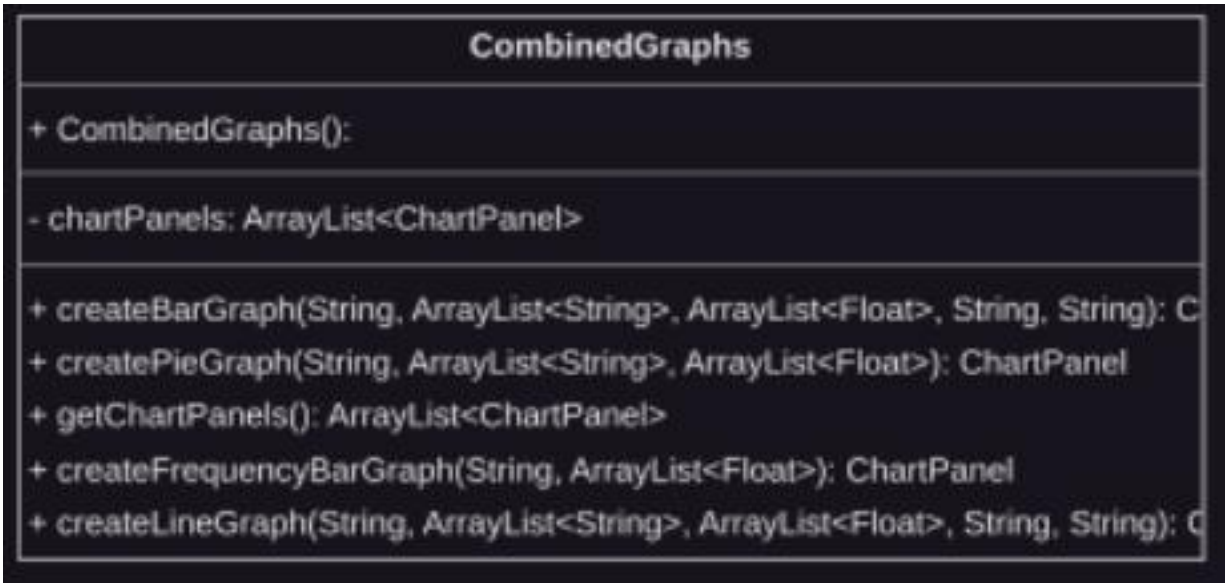
- `createLineGraph(String title, ArrayList<String> ejeX, ArrayList<Float> ejeY, String tituloX, String tituloY)`: Crea un gráfico de líneas con los datos proporcionados y devuelve un `ChartPanel`.

- `createFrequencyBarGraph(String title, ArrayList<Float> numbers)`: Crea un gráfico de barras de frecuencia con los datos proporcionados y devuelve un `ChartPanel`.

- `addChartPanel(ChartPanel chartPanel)`: Añade un `ChartPanel` a la lista de `chartPanels`.

- `getChartPanels()`: Devuelve la lista de `chartPanels`.

La clase `CombinedGraphs` se utiliza para encapsular la creación de diferentes tipos de gráficos en un solo objeto. Esto facilita la creación y gestión de gráficos en la aplicación.



Simbols:

La clase `Simbols` es una clase en Java que utiliza el patrón de diseño Builder para su construcción. Aquí está la descripción de sus atributos y métodos:

Atributos:

- `name`: Un `String` que representa el nombre del símbolo.

- `type`: Un `String` que representa el tipo del símbolo.

- `Svalue`: Un `String` que representa el valor del símbolo en formato de cadena.

- `Fvalue`: Un `float` que representa el valor del símbolo en formato de número flotante.

- `ASvalue`: Un `ArrayList<String>` que representa el valor del símbolo en formato de lista de cadenas.

- `AFvalue`: Un `ArrayList<Float>` que representa el valor del símbolo en formato de lista de números flotantes.

- `line`: Un `int` que representa la línea donde se encuentra el símbolo.

- `column`: Un `int` que representa la columna donde se encuentra el símbolo.

Métodos:

- `Builder`: Es una clase interna que sigue el patrón de diseño Builder. Tiene los mismos atributos que la clase `Simbols` y métodos para establecer estos atributos.

- `getName`, `getType`, `getSvalue`, `getFvalue`, `getASvalue`, `getAFvalue`, `getLine`, `getColumn`: Son métodos getter que devuelven el valor de los atributos correspondientes.

- `setName`, `setType`, `setSvalue`, `setFvalue`, `setASvalue`, `setAFvalue`, `setLine`, `setColumn`: Son métodos setter que establecen el valor de los atributos correspondientes.

- `toString`: Este método sobrescribe el método `toString` de la clase `Object` para proporcionar una representación en cadena de la clase `Simbols`.La clase `Simbols` es una clase

en Java que utiliza el patrón de diseño Builder para su construcción. Aquí está la descripción de sus atributos y métodos:

Atributos:

- `name`: Un `String` que representa el nombre del símbolo.

- `type`: Un `String` que representa el tipo del símbolo.

- `Svalue`: Un `String` que representa el valor del símbolo en formato de cadena.

- `Fvalue`: Un `float` que representa el valor del símbolo en formato de número flotante.

- `ASvalue`: Un `ArrayList<String>` que representa el valor del símbolo en formato de lista de cadenas.

- `AFvalue`: Un `ArrayList<Float>` que representa el valor del símbolo en formato de lista de números flotantes.

- `line`: Un `int` que representa la línea donde se encuentra el símbolo.

- `column`: Un `int` que representa la columna donde se encuentra el símbolo.

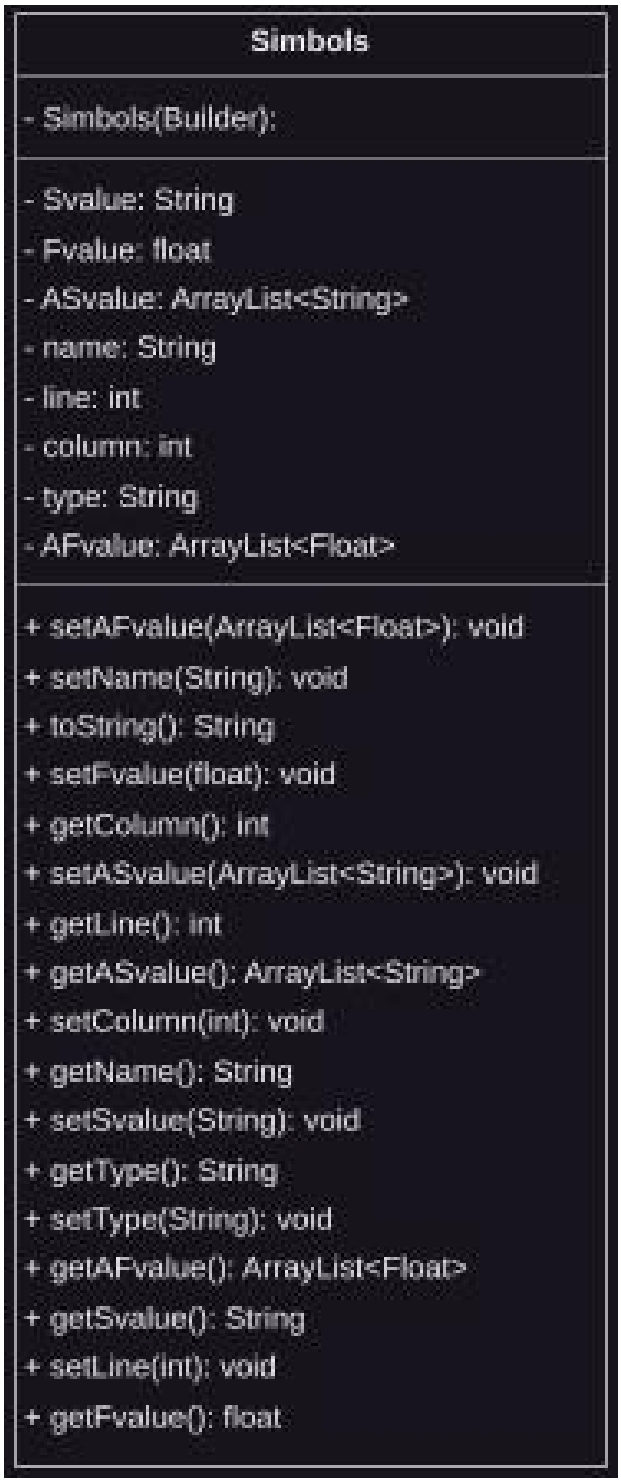
Métodos:

- `Builder`: Es una clase interna que sigue el patrón de diseño Builder. Tiene los mismos atributos que la clase `Simbols` y métodos para establecer estos atributos.

- `getName`, `getType`, `getSvalue`, `getFvalue`, `getASvalue`, `getAFvalue`, `getLine`, `getColumn`: Son métodos getter que devuelven el valor de los atributos correspondientes.

- `setName`, `setType`, `setSvalue`, `setFvalue`, `setASvalue`, `setAFvalue`, `setLine`, `setColumn`: Son métodos setter que establecen el valor de los atributos correspondientes.

- `toString`: Este método sobrescribe el método `toString` de la clase `Object` para proporcionar una representación en cadena de la clase `Simbols`.



Parser:

La clase `Parser` es una clase generada por CUP, que es un generador de analizadores sintácticos LALR para Java. Esta clase se utiliza para analizar la sintaxis de un lenguaje de programación o un formato de datos específico.

La clase `Parser` tiene varios métodos y atributos, algunos de los cuales son:

- `getSymbolContainer()`: Este método devuelve la clase que contiene los símbolos generados por CUP.

- `production_table()`, `action_table()`, `reduce_table()`: Estos métodos devuelven las tablas de producción, acción y reducción respectivamente, que son utilizadas por el analizador sintáctico.

- `do_action()`: Este método se utiliza para invocar una acción de análisis suministrada por el usuario.

- `start_state()`, `start_production()`, `EOF_sym()`, `error_sym()`: Estos métodos devuelven el estado inicial, la producción inicial, el índice del símbolo EOF y el índice del símbolo de error respectivamente.

Además, la clase `Parser` tiene una clase interna `CUP$Parser$actions` que encapsula el código de acción suministrado por el usuario. Esta clase tiene un método `CUP$Parser$do_action_part00000000()` que contiene el código de acción real para las acciones de análisis.

Por último, la clase `Parser` tiene varios atributos que representan las tablas de producción, acción y reducción, así como un objeto de la clase `CUP$Parser$actions` que se utiliza para realizar las acciones de análisis.La clase `Parser` es una clase generada por CUP, que es un generador de analizadores sintácticos LALR para Java. Esta clase se utiliza para analizar la sintaxis de un lenguaje de programación o un formato de datos específico.

La clase `Parser` tiene varios métodos y atributos, algunos de los cuales son:

- `getSymbolContainer()`: Este método devuelve la clase que contiene los símbolos generados por CUP.

- `production_table()`, `action_table()`, `reduce_table()`: Estos métodos devuelven las tablas de producción, acción y reducción respectivamente, que son utilizadas por el analizador sintáctico.

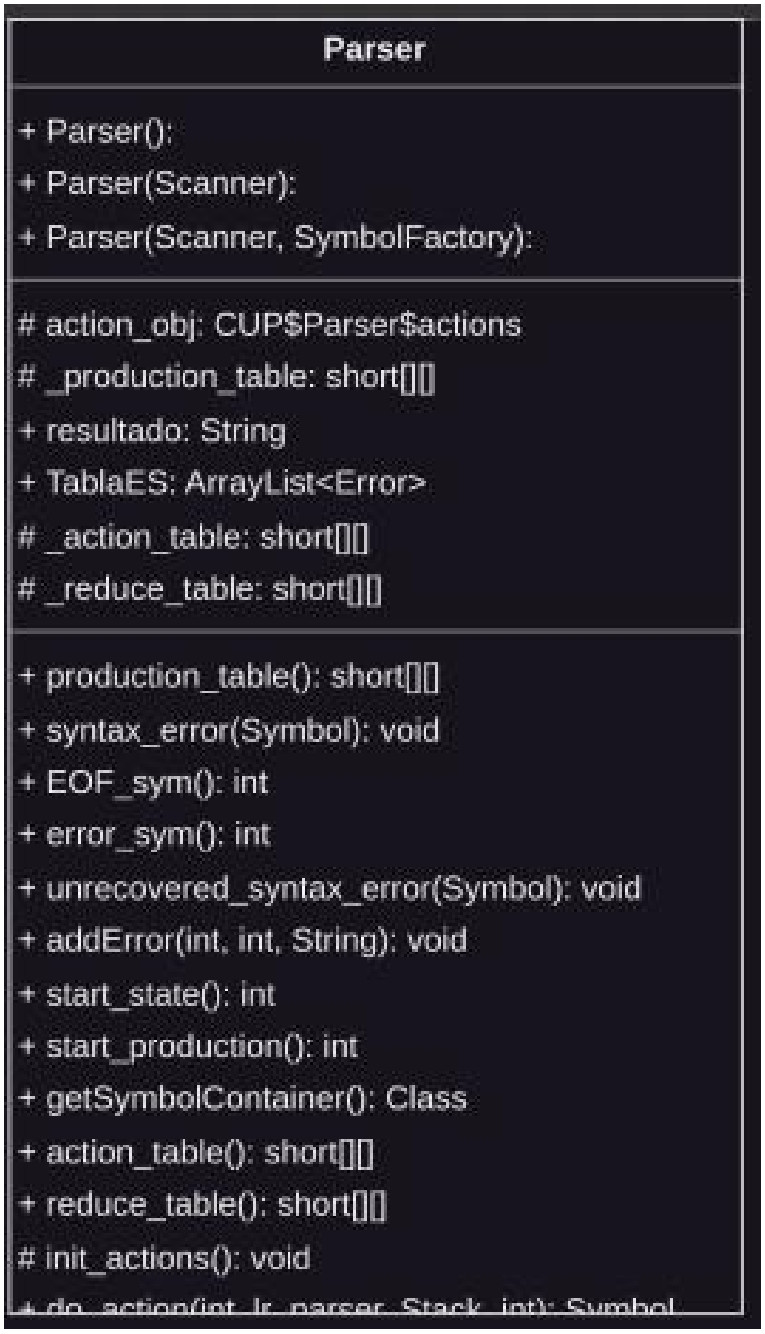
- `do_action()`: Este método se utiliza para invocar una acción de análisis suministrada por el usuario.

- `start_state()`, `start_production()`, `EOF_sym()`, `error_sym()`: Estos métodos devuelven el estado inicial, la producción inicial, el índice del símbolo EOF y el índice del símbolo de error respectivamente.

Además, la clase `Parser` tiene una clase interna `CUP$Parser$actions` que encapsula el código de acción suministrado por el usuario. Esta clase tiene un método

`CUP$Parser$do_action_part00000000()` que contiene el código de acción real para las acciones de análisis.

Por último, la clase `Parser` tiene varios atributos que representan las tablas de producción, acción y reducción, así como un objeto de la clase `CUP$Parser$actions` que se utiliza para realizar las acciones de análisis.



Base_JPanel:

La clase `Base_JPanel` es una subclase de `javax.swing.JPanel` en Java. Esta clase parece ser una parte de una interfaz gráfica de usuario (GUI) que incluye un área de código, una consola y un panel para gráficos. Aquí está la descripción de sus atributos y métodos:

Atributos:

- `filePath`: Un `String` que representa la ruta del archivo.

- `codeTextArea`: Un `String` que representa el área de texto del código.

- `numbers`: Un objeto de la clase `Num_Line_Text_Area` que parece estar relacionado con el área de texto del código.

- `combinedGraphs`: Un objeto de la clase `CombinedGraphs` que parece estar relacionado con los gráficos que se muestran en la GUI.

- `last`: Un `int` que se utiliza en los métodos para navegar a través de los gráficos.

Métodos:

- `Base_JPanel`: Es el constructor de la clase que inicializa los componentes de la GUI y establece los atributos `filePath` y `codeTextArea`.

- `setGraphs`: Este método establece el atributo `combinedGraphs`.

- `returnFilePath`, `returnTextAreaCode`: Estos métodos devuelven los atributos `filePath` y `codeTextArea` respectivamente.

- `setConsoleText`: Este método establece el texto de la consola en la GUI.

- `graphGraph`: Este método actualiza el gráfico que se muestra en la GUI.

- `initComponents`: Este método inicializa los componentes de la GUI.

- `jButtonbeforeMousePressed`, `jButtonafterMousePressed`: Estos métodos se invocan cuando se presionan los botones "Anterior" y "Siguiente" respectivamente. Parecen estar relacionados con la navegación a través de los gráficos.

Además, la clase `Base_JPanel` tiene varios atributos que representan los componentes de la GUI, como botones, etiquetas, paneles y áreas de texto.La clase `Base_JPanel` es una subclase de `javax.swing.JPanel` en Java. Esta clase parece ser una parte de una interfaz gráfica de usuario (GUI) que incluye un área de código, una consola y un panel para gráficos. Aquí está la descripción de sus atributos y métodos:

Atributos:

- `filePath`: Un `String` que representa la ruta del archivo.

- `codeTextArea`: Un `String` que representa el área de texto del código.

- `numbers`: Un objeto de la clase `Num_Line_Text_Area` que parece estar relacionado con el área de texto del código.

- `combinedGraphs`: Un objeto de la clase `CombinedGraphs` que parece estar relacionado con los gráficos que se muestran en la GUI.

- `last`: Un `int` que se utiliza en los métodos para navegar a través de los gráficos.

Métodos:

- `Base_JPanel`: Es el constructor de la clase que inicializa los componentes de la GUI y establece los atributos `filePath` y `codeTextArea`.

- `setGraphs`: Este método establece el atributo `combinedGraphs`.

- `returnFilePath`, `returnTextAreaCode`: Estos métodos devuelven los atributos `filePath` y `codeTextArea` respectivamente.

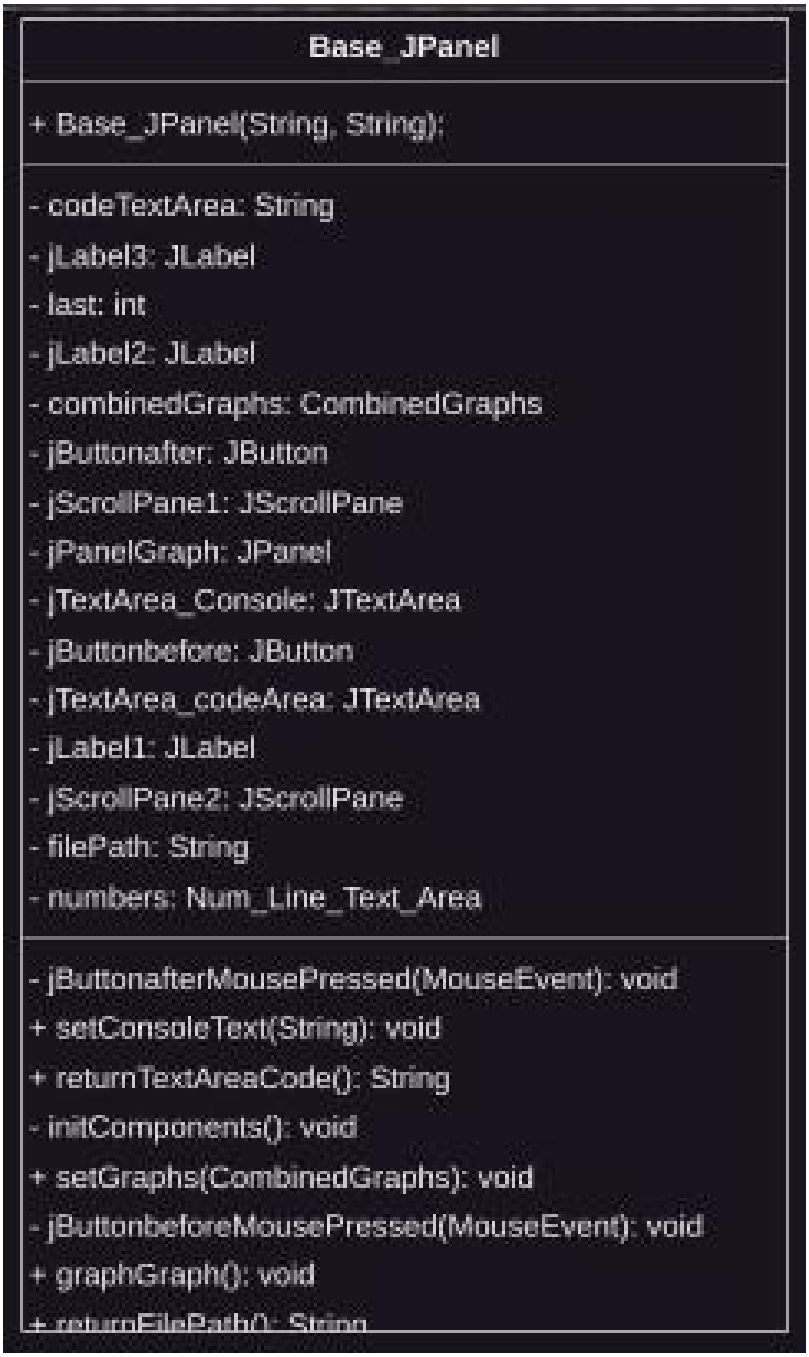
- `setConsoleText`: Este método establece el texto de la consola en la GUI.

- `graphGraph`: Este método actualiza el gráfico que se muestra en la GUI.

- `initComponents`: Este método inicializa los componentes de la GUI.

- `jButtonbeforeMousePressed`, `jButtonafterMousePressed`: Estos métodos se invocan cuando se presionan los botones "Anterior" y "Siguiente" respectivamente. Parecen estar relacionados con la navegación a través de los gráficos.

Además, la clase `Base_JPanel` tiene varios atributos que representan los componentes de la GUI, como botones, etiquetas, paneles y áreas de texto.



Main_JFrame:

La clase `Main_JFrame` es una subclase de `javax.swing.JFrame` en Java. Esta clase parece ser la interfaz gráfica de usuario principal (GUI) de la aplicación. Aquí está la descripción de sus atributos y métodos:

Atributos:

- `jTabbedPane1`: Un objeto de la clase `javax.swing.JTabbedPane` que representa un panel con pestañas donde se pueden agregar y eliminar pestañas.

- `jMenuBar1`, `jMenu_File_CloseFile`, `jMenu_Tabs`, `jMenu_Run`, `jMenu_Reports`: Son objetos de las clases `javax.swing.JMenuBar` y `javax.swing.JMenu` que representan la barra de menú y los menús en la GUI.

- `jMenuItem_NewFile`, `jMenuItem_OpenFile`, `jMenuItem_SaveFile`, `jMenuItem_CloseFile`, `jMenuItem_TokensReport`, `jMenuItem2_ErrorsReport`, `jMenuItem3_SymbolTable`: Son objetos de la clase `javax.swing.JMenuItem` que representan los elementos de menú en la GUI.

Métodos:

- `Main_JFrame`: Es el constructor de la clase que inicializa los componentes de la GUI.

- `initComponents`: Este método inicializa los componentes de la GUI.

- `jMenuItem_NewFileMouseReleased`, `jMenuItem_OpenFileMouseReleased`, `jMenuItem_CloseFileMouseReleased`, `jMenuItem_SaveFileMouseReleased`, `jMenuItem_TokensReportMouseReleased`, `jMenuItem2_ErrorsReportMouseReleased`, `jMenuItem3_SymbolTableMouseReleased`, `jMenu_RunMousePressed`: Estos métodos se invocan cuando se liberan los elementos de menú correspondientes o se presiona el menú "Run". Realizan acciones como crear un nuevo archivo, abrir un archivo, cerrar un archivo, guardar un archivo, generar informes de tokens y errores, y ejecutar el análisis e interpretación del código.

Además, la clase `Main_JFrame` tiene varios atributos que representan los componentes de la GUI, como menús, elementos de menú y un panel con pestañas.La clase `Main_JFrame` es una subclase de `javax.swing.JFrame` en Java. Esta clase parece ser la interfaz gráfica de usuario principal (GUI) de la aplicación. Aquí está la descripción de sus atributos y métodos:

Atributos:

- `jTabbedPane1`: Un objeto de la clase `javax.swing.JTabbedPane` que representa un panel con pestañas donde se pueden agregar y eliminar pestañas.

- `jMenuBar1`, `jMenu_File_CloseFile`, `jMenu_Tabs`, `jMenu_Run`, `jMenu_Reports`: Son objetos de las clases `javax.swing.JMenuBar` y `javax.swing.JMenu` que representan la barra de menú y los menús en la GUI.

- `jMenuItem_NewFile`, `jMenuItem_OpenFile`, `jMenuItem_SaveFile`, `jMenuItem_CloseFile`, `jMenuItem_TokensReport`, `jMenuItem2_ErrorsReport`, `jMenuItem3_SymbolTable`: Son objetos de la clase `javax.swing.JMenuItem` que representan los elementos de menú en la GUI.

Métodos:

- `Main_JFrame`: Es el constructor de la clase que inicializa los componentes de la GUI.

- `initComponents`: Este método inicializa los componentes de la GUI.

- `jMenuItem_NewFileMouseReleased`, `jMenuItem_OpenFileMouseReleased`, `jMenuItem_CloseFileMouseReleased`, `jMenuItem_SaveFileMouseReleased`, `jMenuItem_TokensReportMouseReleased`, `jMenuItem2_ErrorsReportMouseReleased`, `jMenuItem3_SymbolTableMouseReleased`, `jMenu_RunMousePressed`: Estos métodos se invocan

cuando se liberan los elementos de menú correspondientes o se presiona el menú "Run". Realizan acciones como crear un nuevo archivo, abrir un archivo, cerrar un archivo, guardar un archivo, generar informes de tokens y errores, y ejecutar el análisis e interpretación del código.

Además, la clase `Main_JFrame` tiene varios atributos que representan los componentes de la GUI, como menús, elementos de menú y un panel con pestañas.

Interprete:

La clase `Interpreter` es una clase en Java que se utiliza para interpretar un conjunto de instrucciones. Aquí está la descripción de sus atributos y métodos:

Atributos:

- `hash`: Un `HashMap` que mapea una cadena a un objeto `Simbols`. Se utiliza para almacenar las variables y sus valores.

- `Instruccions`: Un `ArrayList` de objetos `Intruccions`. Se utiliza para almacenar las instrucciones que se van a interpretar.

- `combinedGraphs`: Un objeto de la clase `CombinedGraphs`. Se utiliza para crear y almacenar los gráficos que se generan durante la interpretación.

- `console_text`: Un `String` que representa el texto que se imprimirá en la consola.

- `ARITFUNC`: Una lista de cadenas que representa las funciones aritméticas disponibles.

- `ESTAFUNC`: Una lista de cadenas que representa las funciones estadísticas disponibles.

Métodos:

- `Interpreter`: Es el constructor de la clase que inicializa los atributos `Instruccions` y `console_text`.

- `run`: Este método interpreta las instrucciones almacenadas en `Instruccions`.

- `graphPH`, `graphGraphBL`: Estos métodos se utilizan para crear gráficos de pastel y de barras/lineas respectivamente.

- `createFrequencyTable`: Este método crea una tabla de frecuencias a partir de una lista de números.

- `declareArray`, `declareVariable`: Estos métodos se utilizan para declarar una variable o un array respectivamente.

- `isParsableToFloat`: Este método verifica si una cadena se puede convertir en un número flotante.

- `getFloatValue`: Este método obtiene un valor flotante a partir de una cadena.

- `media`, `mediana`, `moda`, `varianza`, `max`, `min`: Estos métodos calculan la media, la mediana, la moda, la varianza, el máximo y el mínimo de una lista de números respectivamente.

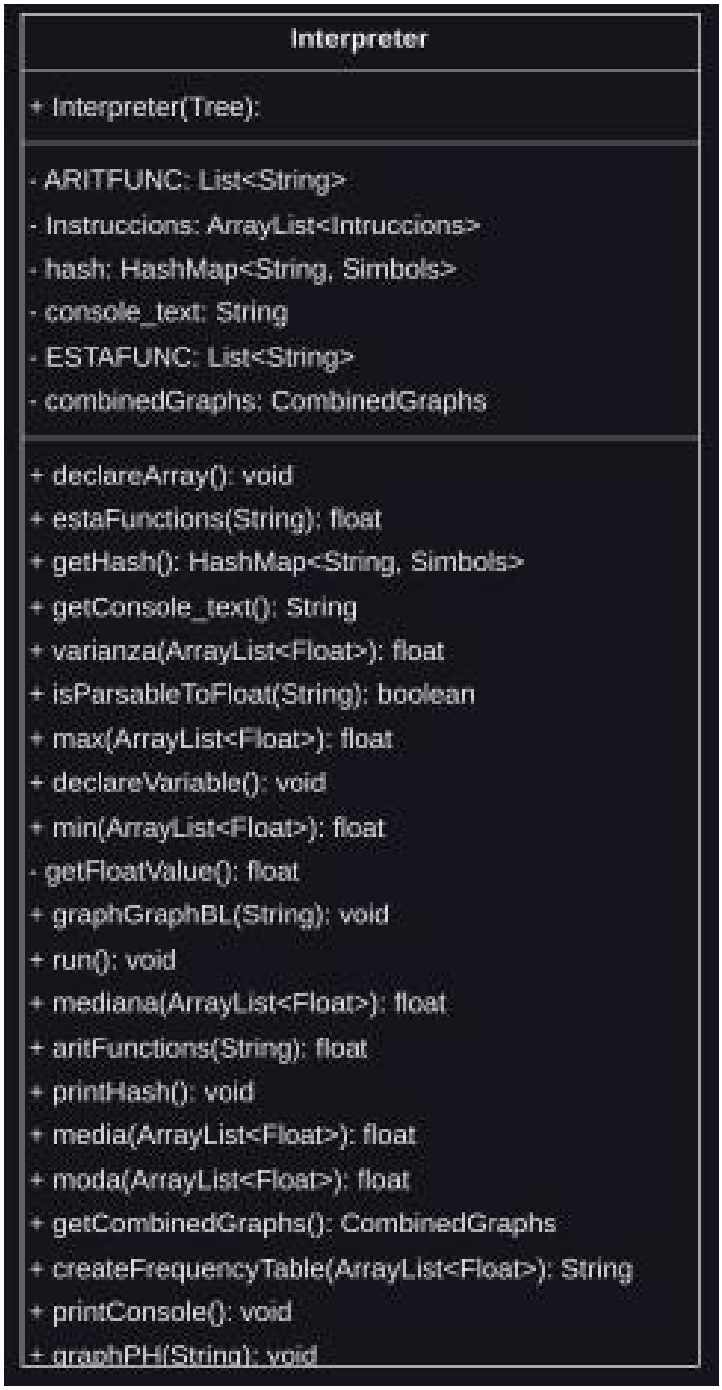
- `estaFunctions`, `aritFunctions`: Estos métodos realizan operaciones estadísticas y aritméticas respectivamente.

- `printConsole`: Este método imprime texto en la consola.

- `printHash`: Este método imprime el contenido del atributo `hash`.

- `getConsole_text`, `getCombinedGraphs`, `getHash`: Son métodos getter que devuelven el valor de los atributos `console_text`, `combinedGraphs` y `hash` respectivamente.

La clase `Interpreter` interpreta un conjunto de instrucciones y realiza operaciones como la declaración de variables, la impresión en la consola, la creación de gráficos y la realización de operaciones aritméticas y estadísticas.



Reportes:

La clase `Reports` es una clase en Java que se utiliza para generar informes en formato HTML. Aquí está la descripción de sus atributos y métodos:

Atributos:

- `errors`: Un `ArrayList<Error>` que almacena los errores que se han producido.

Métodos:

- `tokensReport`: Este método genera un informe de tokens en formato HTML. Recibe una lista de tokens, crea una tabla HTML con los detalles de cada token y escribe el HTML en un archivo llamado "TablaTokens.html".

- `errorsReport`: Este método genera un informe de errores en formato HTML. Recibe una lista de errores, crea una tabla HTML con los detalles de cada error y escribe el HTML en un archivo llamado "TablaErrores.html".

- `simbolTable`: Este método genera un informe de la tabla de símbolos en formato HTML. Recibe un `HashMap` que mapea una cadena a un objeto `Simbols`, crea una tabla HTML con los detalles de cada símbolo y escribe el HTML en un archivo llamado "TablaSimbolos.html".

La clase `Reports` se utiliza para generar informes de tokens, errores y la tabla de símbolos en formato HTML. Estos informes pueden ser útiles para depurar y entender el comportamiento de la aplicación.La clase `Reports` es una clase en Java

que se utiliza para generar informes en formato HTML. Aquí está la descripción de sus atributos y métodos:

Atributos:

- `errors`: Un `ArrayList<Error>` que almacena los errores que se han producido.

Métodos:

- `tokensReport`: Este método genera un informe de tokens en formato HTML. Recibe una lista de tokens, crea una tabla HTML con los detalles de cada token y escribe el HTML en un archivo llamado "TablaTokens.html".

- `errorsReport`: Este método genera un informe de errores en formato HTML. Recibe una lista de errores, crea una tabla HTML con los detalles de cada error y escribe el HTML en un archivo llamado "TablaErrores.html".

- `simbolTable`: Este método genera un informe de la tabla de símbolos en formato HTML. Recibe un `HashMap` que mapea una cadena a un objeto `Simbols`, crea una tabla HTML con los detalles de cada símbolo y escribe el HTML en un archivo llamado "TablaSimbolos.html".

La clase `Reports` se utiliza para generar informes de tokens, errores y la tabla de símbolos en formato HTML. Estos informes pueden ser útiles para depurar y entender el comportamiento de la aplicación.

El lenguaje utilizado fue Java, con jdk17, herramientas utilizadas jflex y cup.