MANUAL TECNICO
ANALIZADOR DE TEXTO

LENGUAJE: JAVA

JDK : 17

FRONTEND:

Clase InterfazGrafica
Descripción General
La clase InterfazGrafica extiende JFrame y se encarga de crear una interfaz gráfica para un analizador de código. Esta interfaz permite al usuario cargar archivos, generar reportes, y visualizar y configurar una cuadrícula.

Componentes Utilizados
JFrame
Descripción: Es una ventana con bordes y botones estándar (minimizar, maximizar, cerrar).
Uso: La clase InterfazGrafica extiende JFrame para crear la ventana principal de la aplicación.
GridPanel
Descripción: Panel personalizado para mostrar una cuadrícula.
Uso: Se utiliza para visualizar y configurar una cuadrícula en la interfaz.
JComboBox
Descripción: Componente que permite seleccionar un elemento de una lista desplegable.
Uso: rowSelector y colSelector permiten seleccionar el número de filas y columnas de la cuadrícula.
JTextArea
Descripción: Área de texto donde el usuario puede escribir o pegar código.
Uso: textArea1 es el área principal donde se ingresa el código a analizar.
JScrollPane
Descripción: Panel con barras de desplazamiento para contener componentes grandes.
Uso: Contiene textArea1 y LineNumberingTextArea para permitir el desplazamiento.
JPanel
Descripción: Contenedor genérico para agrupar otros componentes.
Uso: jPanel1 y selectorPanel agrupan y organizan componentes en la interfaz.
JButton
Descripción: Botón que puede ser clicado para realizar una acción.
Uso: boton y saveButton permiten aceptar configuraciones y guardar imágenes, respectivamente.
JMenuBar
Descripción: Barra de menú que contiene menús desplegables.
Uso: menuBar contiene los menús menuArchivo y menuReporte.
JMenu
Descripción: Menú desplegable que contiene elementos de menú.
Uso: menuArchivo y menuReporte agrupan opciones relacionadas con archivos y reportes.
JMenuItem
Descripción: Elemento de un menú que puede ser seleccionado.
Uso: menuItemCargar y menuItemReporte permiten cargar archivos y generar reportes.
GridBagLayout
Descripción: Administrador de diseño que organiza componentes en una cuadrícula flexible.
Uso: Se utiliza para organizar los componentes en el JFrame.

GridBagConstraints
Descripción: Restricciones para los componentes en un GridBagLayout.
Uso: Define la posición y el comportamiento de los componentes en el GridBagLayout.
FlatLightLaf
Descripción: Look and Feel moderno y plano para aplicaciones Swing.
Uso: Se aplica a la interfaz para mejorar su apariencia visual.
Automata
Descripción: Clase encargada de analizar el código ingresado.
Uso: automata analiza el código y genera tokens.
FileHandler
Descripción: Clase encargada de manejar operaciones de archivo.
Uso: fileHandler carga archivos en textArea1 y guarda imágenes de gridPanel.
ReportGenerator
Descripción: Clase encargada de generar reportes.
Uso: reportGenerator genera reportes basados en el contenido de gridPanel.
Funcionalidad General de la Clase
Configuración de la Interfaz: La clase configura la ventana principal (JFrame) con un GridBagLayout y añade componentes como JTextArea, JScrollPane, JPanel, JComboBox, JButton, y JMenuBar.
Interacción del Usuario: Permite al usuario seleccionar filas y columnas para la cuadrícula, cargar archivos, generar reportes y guardar imágenes.
Análisis de Código: Utiliza la clase Automata para analizar el código ingresado en textArea1 y actualizar la cuadrícula con los tokens generados.
Manejo de Archivos: Utiliza FileHandler para cargar archivos en el área de texto y guardar imágenes de la cuadrícula.
Generación de Reportes: Utiliza ReportGenerator para crear reportes basados en el contenido de la cuadrícula.

Clase ReportGenerator
Descripción General
La clase ReportGenerator se encarga de generar reportes visuales basados en el contenido de un GridPanel. Estos reportes incluyen información detallada sobre los tokens analizados, como su tipo, lexema, posición en la cuadrícula y color asociado.

Componentes Utilizados
GridPanel
Descripción: Panel personalizado para mostrar una cuadrícula.
Uso: gridPanel se utiliza para obtener los componentes que contienen la información de los tokens.
Map
Descripción: Estructura de datos que asocia claves con valores.
Uso: colorToTokenTypeMap mapea colores a tipos de tokens.
JTable
Descripción: Componente que muestra datos en una tabla.
Uso: table muestra los datos de los tokens en un formato tabular.
DefaultTableModel

Descripción: Modelo de datos por defecto para JTable.
Uso: model gestiona los datos y la estructura de la tabla.
JScrollPane
Descripción: Panel con barras de desplazamiento para contener componentes grandes.
Uso: scrollPane contiene la tabla para permitir el desplazamiento.
JFrame
Descripción: Ventana con bordes y botones estándar (minimizar, maximizar, cerrar).
Uso: reportFrame es la ventana que muestra el reporte de tokens.
TableCellRenderer
Descripción: Interfaz que define cómo se renderizan las celdas de una tabla.
Uso: JLabelRenderer personaliza la renderización de las celdas que contienen JLabel.
JLabel
Descripción: Componente que muestra texto o imágenes.
Uso: cuadroLabel muestra información detallada sobre cada token en la tabla.
BorderLayout
Descripción: Administrador de diseño que organiza componentes en cinco regiones: norte, sur, este, oeste y centro.
Uso: Se utiliza para organizar los componentes dentro de reportFrame.
Funcionalidad General de la Clase
Inicialización: El constructor ReportGenerator inicializa el GridPanel y el mapa colorToTokenTypeMap con los colores y tipos de tokens correspondientes.
Generación de Reportes: El método generateReport crea un reporte visual en una nueva ventana (JFrame). Este reporte incluye una tabla (JTable) que muestra los tokens con su tipo, lexema, posición y color.
Renderización de Celdas: La clase interna JLabelRenderer implementa TableCellRenderer para personalizar la renderización de las celdas que contienen JLabel en la tabla.
Interfaz de Usuario: La tabla se muestra dentro de un JScrollPane para permitir el desplazamiento, y todo el contenido se organiza en un JFrame con un BorderLayout.

Clase LineNumberingTextArea
Descripción General
La clase LineNumberingTextArea extiende JTextArea y se utiliza para mostrar números de línea junto a un JTextArea principal. Esto es útil para editores de texto y entornos de desarrollo donde es importante visualizar el número de cada línea de código.

Componentes Utilizados
JTextArea
Descripción: Área de texto donde el usuario puede escribir o pegar código.
Uso: textArea es el área de texto principal a la que se le añaden los números de línea.
DocumentListener
Descripción: Interfaz que escucha cambios en el documento de un JTextComponent.
Uso: Se utiliza para actualizar los números de línea cada vez que el contenido del textArea cambia.
DocumentEvent
Descripción: Evento que representa un cambio en el documento.

Uso: Se pasa a los métodos del DocumentListener para manejar las actualizaciones del documento.
Funcionalidad General de la Clase
Inicialización: El constructor LineNumberingTextArea toma un JTextArea como parámetro y añade un DocumentListener a su documento para escuchar cambios.
Actualización de Números de Línea: El método updateLineNumbers se encarga de generar y actualizar los números de línea cada vez que el contenido del textArea cambia. Recorre todas las líneas del textArea y construye una cadena con los números de línea correspondientes.
Sincronización con el JTextArea Principal: Cada vez que se inserta, elimina o cambia el contenido del textArea, el DocumentListener llama a updateLineNumbers para mantener los números de línea sincronizados con el contenido actual.

Clase FileHandler
Descripción General
La clase FileHandler se encarga de manejar operaciones de archivo, como cargar archivos de texto en un JTextArea y guardar imágenes de un JPanel en formatos PNG o JPG.

Componentes Utilizados
JFileChooser
Descripción: Componente que permite al usuario elegir archivos y directorios.
Uso: fileChooser se utiliza para seleccionar archivos a cargar y para elegir la ubicación donde guardar imágenes.
FileNameExtensionFilter
Descripción: Filtro que permite mostrar solo archivos con ciertas extensiones en el JFileChooser.
Uso: Se utiliza para filtrar archivos de texto (txt) al cargar y archivos de imagen (png, jpg) al guardar.
BufferedReader
Descripción: Clase que lee texto de una entrada de caracteres de manera eficiente.
Uso: br se utiliza para leer el contenido del archivo de texto seleccionado.
FileReader
Descripción: Clase que lee archivos de texto.
Uso: FileReader se utiliza junto con BufferedReader para leer el archivo de texto seleccionado.
BufferedImage
Descripción: Clase que representa una imagen en memoria.
Uso: image se utiliza para crear una imagen de la representación gráfica del JPanel.
Graphics2D
Descripción: Clase que extiende Graphics para proporcionar un control más sofisticado sobre la geometría, la transformación de coordenadas, la gestión del color y el estilo de renderizado.
Uso: g2d se utiliza para dibujar la representación gráfica del JPanel en el BufferedImage.
ImageIO
Descripción: Clase que contiene métodos para leer y escribir imágenes.

Uso: ImageIO.write se utiliza para guardar la imagen en el archivo seleccionado.
JOptionPane
Descripción: Clase que proporciona métodos estándar para mostrar cuadros de diálogo.
Uso: JOptionPane.showMessageDialog se utiliza para mostrar mensajes de éxito o error al guardar la imagen.
Funcionalidad General de la Clase
Cargar Archivo: El método cargarArchivo permite al usuario seleccionar un archivo de texto mediante un JFileChooser. Si se selecciona un archivo, su contenido se lee y se carga en el JTextArea proporcionado.
Guardar Imagen: El método guardarImagen permite al usuario guardar una imagen de un JPanel en formato PNG o JPG. Se crea un BufferedImage de la representación gráfica del JPanel, y se guarda en el archivo seleccionado utilizando ImageIO.write.
Interacción del Usuario: Utiliza JFileChooser para seleccionar archivos y JOptionPane para mostrar mensajes de confirmación o error.

Clase GridPanel
Descripción General
La clase GridPanel extiende JPanel y se utiliza para crear y gestionar una cuadrícula de celdas. Cada celda puede contener información sobre tokens, incluyendo su color y posición en la cuadrícula.

Componentes Utilizados
GridLayout
Descripción: Administrador de diseño que organiza los componentes en una cuadrícula con un número fijo de filas y columnas.
Uso: Se utiliza para organizar las celdas dentro del GridPanel.
BorderFactory
Descripción: Clase que proporciona métodos para crear bordes.
Uso: BorderFactory.createLineBorder se utiliza para crear un borde negro alrededor de cada celda.
MouseAdapter
Descripción: Clase adaptadora que recibe eventos de ratón.
Uso: Se utiliza para manejar eventos de clic en las celdas de la cuadrícula.
MouseEvent
Descripción: Evento que indica una acción del ratón.
Uso: Se pasa al método mouseClicked para obtener información sobre el clic del ratón.
JOptionPane
Descripción: Clase que proporciona métodos estándar para mostrar cuadros de diálogo.
Uso: JOptionPane.showMessageDialog se utiliza para mostrar información sobre el token cuando se hace clic en una celda.
Funcionalidad General de la Clase
Configuración de la Cuadrícula: El método configureGrid establece el número de filas y columnas de la cuadrícula, elimina todos los componentes existentes, y añade nuevas celdas (JPanel) con bordes y propiedades de cliente para la fila y columna.

Actualización de la Cuadrícula: El método updateGrid actualiza las celdas de la cuadrícula con información sobre los tokens, incluyendo su color y texto. También maneja tokens especiales que tienen posiciones específicas en la cuadrícula.
Interacción del Usuario: Cada celda tiene un MouseListener que muestra un cuadro de diálogo con información sobre el token cuando se hace clic en la celda.

BACKEND:

Clase Automata
Descripción General
La clase Automata se encarga de analizar el código ingresado y generar una lista de tokens. También maneja tokens especiales que tienen posiciones específicas en la cuadrícula.

Componentes Utilizados
StringTokenizer
Descripción: Clase que divide una cadena en tokens.
Uso: tokenizer se utiliza para dividir cada línea de código en palabras individuales.
TokenInfo
Descripción: Clase que contiene información sobre un token, incluyendo su texto y color.
Uso: Se utiliza para almacenar y gestionar información sobre cada token analizado.
Funcionalidad General de la Clase
Análisis de Código: El método analizarCodigo toma una cadena de código y un mapa de tokens especiales, y devuelve una lista de TokenInfo. Divide el código en líneas y luego en palabras, ignorando líneas y tokens vacíos. Verifica si una palabra es un token especial y, si es así, la registra en el mapa de tokens especiales. Si no, determina el color del token y lo añade a la lista de TokenInfo.
Verificación de Tokens Especiales: El método esTokenConPosicion verifica si una palabra es un token especial con una posición específica en la cuadrícula.
Registro de Tokens Especiales: El método registrarTokenEspecial añade un token especial al mapa de tokens especiales, utilizando la fila y columna como clave y el color como valor.
Métodos
analizarCodigo
Descripción: Analiza el código ingresado y genera una lista de tokens.
Parámetros:
codigo: Cadena de texto que contiene el código a analizar.
tokenEspecialMap: Mapa que almacena tokens especiales con sus posiciones.
Retorno: Lista de TokenInfo con los tokens analizados.

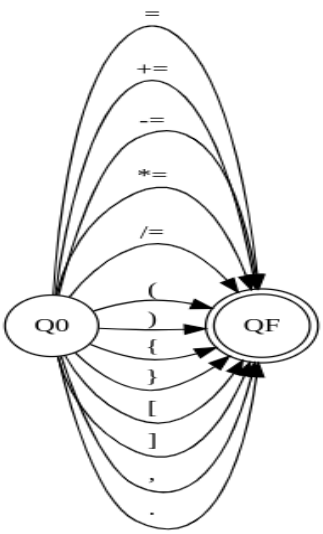
Funcionamiento:
Divide el código en líneas y luego en palabras.
Ignora líneas y tokens vacíos.
Verifica si una palabra es un token especial y la registra en el mapa de tokens especiales.
Determina el color del token y lo añade a la lista de TokenInfo.
esTokenConPosicion
Descripción: Verifica si una palabra es un token especial con una posición específica.
Parámetros:
palabra: Cadena de texto que representa el token a verificar.
Retorno: true si la palabra es un token especial con posición, false en caso contrario.
Funcionamiento:
Busca los paréntesis y las comas en la palabra.
Verifica si hay exactamente tres partes: color, fila y columna.
registrarTokenEspecial
Descripción: Añade un token especial al mapa de tokens especiales.
Parámetros:
palabra: Cadena de texto que representa el token especial.
tokenEspecialMap: Mapa que almacena tokens especiales con sus posiciones.
Funcionamiento:
Extrae el color, fila y columna de la palabra.
Crea una clave única basada en fila y columna.
Almacena el color con la clave en el mapa.

Clase AutomataAsignacionYSignos
Descripción General
La clase AutomataAsignacionYSignos se encarga de validar cadenas que representan operadores de asignación y signos, devolviendo un color asociado a cada tipo de operador o signo.

Componentes Utilizados
Estado
Descripción: Enum o clase que define los diferentes estados del autómata.
Uso: estado se utiliza para manejar los estados del autómata durante la validación de las cadenas.
Funcionalidad General de la Clase
Validación de Asignación y Signos: El método validarAsignacionYSignos toma una cadena y verifica si es un operador de asignación o un signo. Si la cadena es válida, devuelve un color asociado; de lo contrario, devuelve null.
Métodos
validarAsignacionYSignos
Descripción: Valida si una cadena es un operador de asignación o un signo y devuelve el color asociado.
Parámetros:
cadena: Cadena de texto que representa el operador o signo a validar.
Retorno: Cadena de texto que representa el color asociado al operador o signo, o null si la cadena no es válida.

Funcionamiento:
Verifica si la cadena está vacía y retorna null si es así.
Inicializa el estado actual del autómata a Q0.
Utiliza una estructura switch para verificar la cadena y asignar el estado final QF y el color correspondiente según el tipo de operador o signo.
Devuelve el color asociado al operador o signo si es válido, o null si no lo es.

Q: {Q0, QF}
A: {=, +=, -=, *=, /=, (,), {, }, [,], , .}
δ:
δ(Q0, =) = QF
δ(Q0, +=) = QF
δ(Q0, -=) = QF
δ(Q0, *=) = QF
δ(Q0, /=) = QF
δ(Q0, () = QF
δ(Q0,)) = QF
δ(Q0, {) = QF
δ(Q0, }) = QF
δ(Q0, [) = QF
δ(Q0,]) = QF
δ(Q0, ,) = QF
δ(Q0, .) = QF
q_0: Q0
F: {QF}



Clase AutomataIdentificadores

Descripción General

La clase AutomataIdentificadores se encarga de validar si una cadena de texto es un identificador válido según las reglas definidas por el autómata. Un identificador válido debe comenzar con una letra y puede contener letras, dígitos y guiones bajos (_).

Componentes Utilizados

Estado

Descripción: Enum o clase que define los diferentes estados del autómata.

Uso: estado se utiliza para manejar los estados del autómata durante la validación de las cadenas.

Character

Descripción: Clase que contiene métodos estáticos para manipular caracteres.

Uso: Character.isLetter y Character.isDigit se utilizan para verificar si un carácter es una letra o un dígito.

Funcionalidad General de la Clase

Validación de Cadenas: El método validarCadena toma una cadena y verifica si es un identificador válido. Si la cadena es válida, devuelve un color asociado; de lo contrario, devuelve null.

Métodos

validarCadena

Descripción: Valida si una cadena es un identificador válido y devuelve el color asociado.

Parámetros:

cadena: Cadena de texto que representa el identificador a validar.

Retorno: Cadena de texto que representa el color asociado al identificador, o null si la cadena no es válida.

Funcionamiento:

Verifica si la cadena está vacía y retorna null si es así.

Inicializa el estado actual del autómata a Q0.

Verifica el primer carácter de la cadena. Si es una letra, cambia el estado a Q1; de lo contrario, retorna null.

Procesa el resto de los caracteres de la cadena:

Si el estado actual es Q1 y el carácter es una letra, un dígito o un guion bajo, cambia el estado a QF.

Si el estado actual es QF y el carácter es una letra, un dígito o un guion bajo, mantiene el estado en QF.

Si el carácter no cumple con las condiciones anteriores, retorna null.

Al finalizar el procesamiento de la cadena, retorna #FFD300 si el estado final es Q1 o QF; de lo contrario, retorna null.

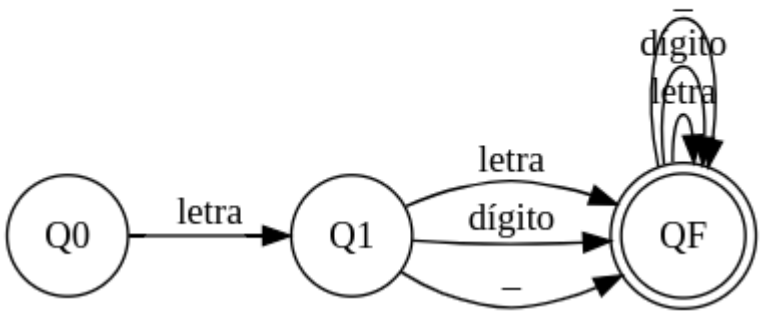
El autómata se puede describir con la siguiente gramática:

Q: {Q0, Q1, QF}

A: {letras, dígitos, _}

δ:

δ(Q0, letra) = Q1
δ(Q1, letra) = QF
δ(Q1, dígito) = QF
δ(Q1, _) = QF
δ(QF, letra) = QF
δ(QF, dígito) = QF
δ(QF, _) = QF
q_0: Q0
F: {QF}



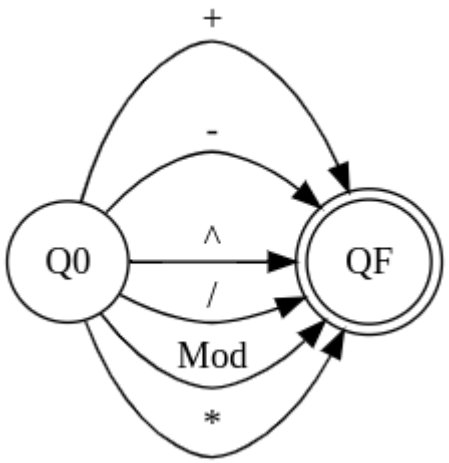
Clase AutomataOperadoresAritmeticos
Descripción General
La clase AutomataOperadoresAritmeticos se encarga de validar si una cadena de texto es un operador aritmético válido según las reglas definidas por el autómata. Un operador aritmético válido puede ser uno de los siguientes: +, -, ^, /, Mod, *.

Componentes Utilizados
Estado
Descripción: Enum o clase que define los diferentes estados del autómata.
Uso: estado se utiliza para manejar los estados del autómata durante la validación de las cadenas.
Funcionalidad General de la Clase
Validación de Operadores Aritméticos: El método validarOperadorAritmetico toma una cadena y verifica si es un operador aritmético válido. Si la cadena es válida, devuelve un color asociado; de lo contrario, devuelve null.
Métodos
validarOperadorAritmetico
Descripción: Valida si una cadena es un operador aritmético válido y devuelve el color asociado.
Parámetros:
cadena: Cadena de texto que representa el operador aritmético a validar.
Retorno: Cadena de texto que representa el color asociado al operador aritmético, o null si la cadena no es válida.
Funcionamiento:

Verifica si la cadena está vacía y retorna null si es así.
Inicializa el estado actual del autómata a Q0.
Utiliza una estructura switch para verificar la cadena y asignar el estado final QF y el color correspondiente según el tipo de operador aritmético.
Devuelve el color asociado al operador aritmético si es válido, o null si no lo es.
Detalles del Funcionamiento
Estado Inicial: El estado inicial del autómata es Q0.
Transiciones de Estado:
Si la cadena es +, -, ^, /, Mod o *, el estado cambia a QF y se devuelve el color correspondiente.
Si la cadena no coincide con ninguno de los operadores aritméticos válidos, se devuelve null.

El autómata se puede describir con la siguiente gramática:

Q: {Q0, QF}
A: {+, -, ^, /, Mod, *}
δ:
δ(Q0, +) = QF
δ(Q0, -) = QF
δ(Q0, ^) = QF
δ(Q0, /) = QF
δ(Q0, Mod) = QF
δ(Q0, *) = QF
q_0: Q0
F: {QF}



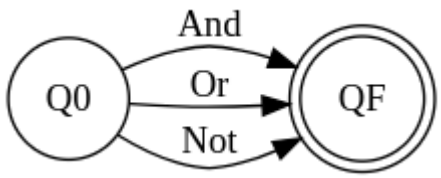
Clase AutomataOperadoresLogicos

Descripción General
La clase AutomataOperadoresLogicos se encarga de validar si una cadena de texto es un operador lógico válido según las reglas definidas por el autómata. Un operador lógico válido puede ser uno de los siguientes: And, Or, Not.

Componentes Utilizados
Estado
Descripción: Enum o clase que define los diferentes estados del autómata.
Uso: estado se utiliza para manejar los estados del autómata durante la validación de las cadenas.
Funcionalidad General de la Clase
Validación de Operadores Lógicos: El método validarOperadoresLogicos toma una cadena y verifica si es un operador lógico válido. Si la cadena es válida, devuelve un color asociado; de lo contrario, devuelve null.
Métodos
validarOperadoresLogicos
Descripción: Valida si una cadena es un operador lógico válido y devuelve el color asociado.
Parámetros:
cadena: Cadena de texto que representa el operador lógico a validar.
Retorno: Cadena de texto que representa el color asociado al operador lógico, o null si la cadena no es válida.
Funcionamiento:
Verifica si la cadena está vacía y retorna null si es así.
Inicializa el estado actual del autómata a Q0.
Utiliza una estructura switch para verificar la cadena y asignar el estado final QF y el color correspondiente según el tipo de operador lógico.
Devuelve el color asociado al operador lógico si es válido, o null si no lo es.
Detalles del Funcionamiento
Estado Inicial: El estado inicial del autómata es Q0.
Transiciones de Estado:
Si la cadena es And, Or o Not, el estado cambia a QF y se devuelve el color correspondiente.
Si la cadena no coincide con ninguno de los operadores lógicos válidos, se devuelve null.

El autómata se puede describir con la siguiente gramática:

Q: {Q0, QF}
A: {And, Or, Not}
δ:
δ(Q0, And) = QF
δ(Q0, Or) = QF
δ(Q0, Not) = QF
q_0: Q0
F: {QF}



Clase AutomataPalabrasReservadas

Descripción General

La clase AutomataPalabrasReservadas se encarga de validar si una cadena de texto es una palabra reservada válida según las reglas definidas por el autómata. Una palabra reservada válida puede ser una de las siguientes: Module, End, Sub, Main, Dim, As, Integer, String, Boolean, Double, Char, Console.WriteLine, Console.ReadLine, If, ElseIf, Else, Then, While, Do, Loop, For, To, Next, Function, Return, Const.

Componentes Utilizados

Estado

Descripción: Enum o clase que define los diferentes estados del autómata.

Uso: estado se utiliza para manejar los estados del autómata durante la validación de las cadenas.

Funcionalidad General de la Clase

Validación de Palabras Reservadas: El método validarPalabraReservada toma una cadena y verifica si es una palabra reservada válida. Si la cadena es válida, devuelve un color asociado; de lo contrario, devuelve null.

Métodos

validarPalabraReservada

Descripción: Valida si una cadena es una palabra reservada válida y devuelve el color asociado.

Parámetros:

cadena: Cadena de texto que representa la palabra reservada a validar.

Retorno: Cadena de texto que representa el color asociado a la palabra reservada, o null si la cadena no es válida.

Funcionamiento:

Verifica si la cadena está vacía y retorna null si es así.

Inicializa el estado actual del autómata a Q0.

Utiliza una estructura switch para verificar la cadena y asignar el estado final QF si la cadena coincide con una palabra reservada válida.

Devuelve el color asociado a la palabra reservada si es válida, o null si no lo es.

Detalles del Funcionamiento

Estado Inicial: El estado inicial del autómata es Q0.

Transiciones de Estado:

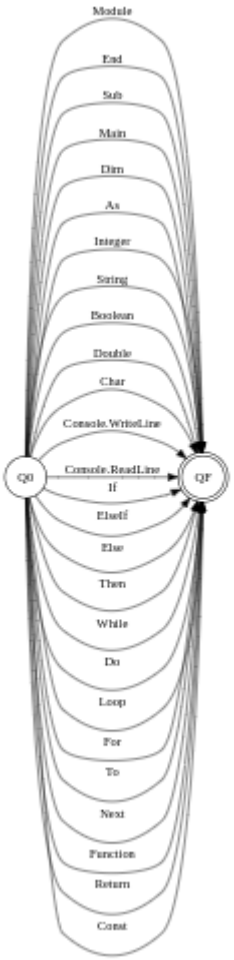
Si la cadena coincide con una de las palabras reservadas válidas, el estado cambia a QF.

Si la cadena no coincide con ninguna de las palabras reservadas válidas, se devuelve null.

Salida: Devuelve #60A917 si la cadena termina en el estado final QF; de lo contrario, devuelve null.

El autómata se puede describir con la siguiente gramática:

Q: {Q0, QF}
A: {Module, End, Sub, Main, Dim, As, Integer, String, Boolean, Double, Char, Console.WriteLine, Console.ReadLine, If, ElseIf, Else, Then, While, Do, Loop, For, To, Next, Function, Return, Const}
δ:
δ(Q0, Module) = QF
δ(Q0, End) = QF
δ(Q0, Sub) = QF
δ(Q0, Main) = QF
δ(Q0, Dim) = QF
δ(Q0, As) = QF
δ(Q0, Integer) = QF
δ(Q0, String) = QF
δ(Q0, Boolean) = QF
δ(Q0, Double) = QF
δ(Q0, Char) = QF
δ(Q0, Console.WriteLine) = QF
δ(Q0, Console.ReadLine) = QF
δ(Q0, If) = QF
δ(Q0, ElseIf) = QF
δ(Q0, Else) = QF
δ(Q0, Then) = QF
δ(Q0, While) = QF
δ(Q0, Do) = QF
δ(Q0, Loop) = QF
δ(Q0, For) = QF
δ(Q0, To) = QF
δ(Q0, Next) = QF
δ(Q0, Function) = QF
δ(Q0, Return) = QF
δ(Q0, Const) = QF
q_0: Q0
F: {QF}



Clase AutomataRelacionalesComparacion

Descripción General

La clase AutomataRelacionalesComparacion se encarga de validar si una cadena de texto es un operador relacional o de comparación válido según las reglas definidas por el autómata. Un operador relacional o de comparación válido puede ser uno de los siguientes: ==, <>, >, <, >=, <=.

Componentes Utilizados

Estado

Descripción: Enum o clase que define los diferentes estados del autómata.

Uso: estado se utiliza para manejar los estados del autómata durante la validación de las cadenas.

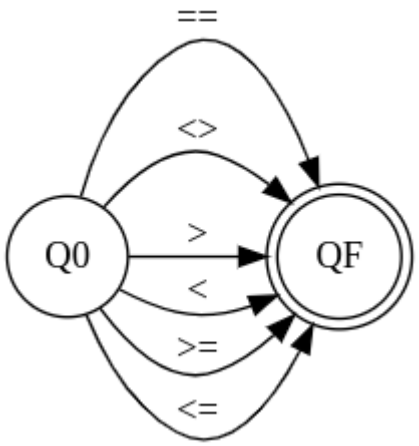
Funcionalidad General de la Clase

Validación de Operadores Relacionales y de Comparación: El método validarOperadorRelacionalComparacion toma una cadena y verifica si es un operador

relacional o de comparación válido. Si la cadena es válida, devuelve un color asociado; de lo contrario, devuelve null.
Métodos
validarOperadorRelacionalComparacion
Descripción: Valida si una cadena es un operador relacional o de comparación válido y devuelve el color asociado.
Parámetros:
cadena: Cadena de texto que representa el operador relacional o de comparación a validar.
Retorno: Cadena de texto que representa el color asociado al operador relacional o de comparación, o null si la cadena no es válida.
Funcionamiento:
Verifica si la cadena está vacía y retorna null si es así.
Inicializa el estado actual del autómata a Q0.
Utiliza una estructura switch para verificar la cadena y asignar el estado final QF y el color correspondiente según el tipo de operador relacional o de comparación.
Devuelve el color asociado al operador relacional o de comparación si es válido, o null si no lo es.
Detalles del Funcionamiento
Estado Inicial: El estado inicial del autómata es Q0.
Transiciones de Estado:
Si la cadena es ==, <>, >, <, >= o <=, el estado cambia a QF y se devuelve el color correspondiente.
Si la cadena no coincide con ninguno de los operadores relacionales o de comparación válidos, se devuelve null.

El autómata se puede describir con la siguiente gramática:

Q: {Q0, QF}
A: {==, <>, >, <, >=, <=}
δ:
δ(Q0, ==) = QF
δ(Q0, <>) = QF
δ(Q0, >) = QF
δ(Q0, <) = QF
δ(Q0, >=) = QF
δ(Q0, <=) = QF
q_0: Q0
F: {QF}



Clase AutomataTipoDeDatos

Descripción General

La clase AutomataTipoDeDatos se encarga de validar si una cadena de texto representa un tipo de dato válido según las reglas definidas por el autómata. Los tipos de datos válidos incluyen números enteros, números decimales, booleanos, cadenas y caracteres.

Componentes Utilizados

Estado

Descripción: Enum o clase que define los diferentes estados del autómata.

Uso: estado se utiliza para manejar los estados del autómata durante la validación de las cadenas.

Integer

Descripción: Clase que contiene métodos estáticos para manipular números enteros.

Uso: Integer.parseInt se utiliza para verificar si una cadena es un número entero.

Double

Descripción: Clase que contiene métodos estáticos para manipular números decimales.

Uso: Double.parseDouble se utiliza para verificar si una cadena es un número decimal.

Character

Descripción: Clase que contiene métodos estáticos para manipular caracteres.

Uso: Character se utiliza para verificar si una cadena es un carácter.

Funcionalidad General de la Clase

Validación de Tipos de Datos: El método validarTipoDato toma una cadena y verifica si representa un tipo de dato válido. Si la cadena es válida, devuelve un color asociado; de lo contrario, devuelve null.

Métodos

validarTipoDato

Descripción: Valida si una cadena representa un tipo de dato válido y devuelve el color asociado.

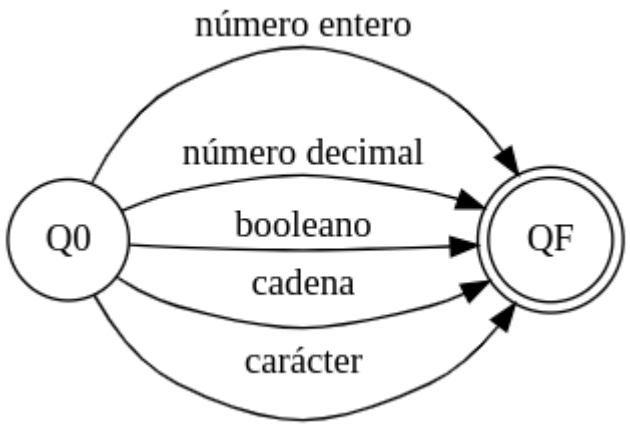
Parámetros:

cadena: Cadena de texto que representa el tipo de dato a validar.

Retorno: Cadena de texto que representa el color asociado al tipo de dato, o null si la cadena no es válida.
Funcionamiento:
Verifica si la cadena está vacía y retorna null si es así.
Inicializa el estado actual del autómata a Q0.
Verificación de Número Entero:
Intenta convertir la cadena a un número entero utilizando Integer.parseInt.
Si la conversión es exitosa, cambia el estado a QF y devuelve el color #1BA1E2.
Si ocurre una NumberFormatException, la cadena no es un número entero.
Verificación de Número Decimal:
Intenta convertir la cadena a un número decimal utilizando Double.parseDouble.
Si la conversión es exitosa y la cadena contiene un punto decimal, cambia el estado a QF y devuelve el color #FFFF88.
Si ocurre una NumberFormatException, la cadena no es un número decimal.
Verificación de Booleano:
Verifica si la cadena es "True" o "False".
Si es así, cambia el estado a QF y devuelve el color #FA6800.
Verificación de Cadena:
Verifica si la cadena comienza y termina con comillas dobles (").
Si es así, cambia el estado a QF y devuelve el color #E51400.
Verificación de Carácter:
Verifica si la cadena comienza y termina con comillas simples (') y tiene una longitud de 3 caracteres.
Si es así, cambia el estado a QF y devuelve el color #0050EF.
Si la cadena no cumple con ninguna de las condiciones anteriores, retorna null.
Detalles del Funcionamiento
Estado Inicial: El estado inicial del autómata es Q0.
Transiciones de Estado:
Si la cadena es un número entero, número decimal, booleano, cadena o carácter válido, el estado cambia a QF y se devuelve el color correspondiente.
Si la cadena no coincide con ninguno de los tipos de datos válidos, se devuelve null.

El autómata se puede describir con la siguiente gramática:

Q: {Q0, QF}
A: {número entero, número decimal, booleano, cadena, carácter}
δ:
δ(Q0, número entero) = QF
δ(Q0, número decimal) = QF
δ(Q0, booleano) = QF
δ(Q0, cadena) = QF
δ(Q0, carácter) = QF
q_0: Q0
F: {QF}



Clase TokenInfo

Descripción General

La clase TokenInfo se utiliza para almacenar y gestionar información sobre tokens. Un token puede representar una palabra, un símbolo, o cualquier otro elemento léxico en el análisis de código. La clase incluye información sobre el texto del token, su posición en términos de fila y columna, y su color asociado.

Componentes Utilizados

String

Descripción: Clase que representa cadenas de caracteres.

Uso: text y color se utilizan para almacenar el texto del token y su color asociado.

int

Descripción: Tipo de dato primitivo que representa números enteros.

Uso: fila y columna se utilizan para almacenar la posición del token en términos de fila y columna.

Funcionalidad General de la Clase

Almacenamiento de Información del Token: La clase TokenInfo almacena el texto del token, su posición (fila y columna) y su color.

Acceso y Modificación de Datos: Proporciona métodos para obtener y establecer estos valores.

Constructores

TokenInfo(String text, String color)

Descripción: Constructor que inicializa un objeto TokenInfo con el texto del token y su color.

Parámetros:

text: Cadena de texto que representa el texto del token.

color: Cadena de texto que representa el color asociado al token.

TokenInfo(String text, int fila, int columna, String color)

Descripción: Constructor que inicializa un objeto TokenInfo con el texto del token, su posición (fila y columna) y su color.

Parámetros:
text: Cadena de texto que representa el texto del token.
fila: Entero que representa la fila en la que se encuentra el token.
columna: Entero que representa la columna en la que se encuentra el token.
color: Cadena de texto que representa el color asociado al token.
Métodos
getText
Descripción: Devuelve el texto del token.
Retorno: Cadena de texto que representa el texto del token.
getFila
Descripción: Devuelve la fila en la que se encuentra el token.
Retorno: Entero que representa la fila del token.
getColumna
Descripción: Devuelve la columna en la que se encuentra el token.
Retorno: Entero que representa la columna del token.
getColor
Descripción: Devuelve el color asociado al token.
Retorno: Cadena de texto que representa el color del token.
setText
Descripción: Establece el texto del token.
Parámetros:
text: Cadena de texto que representa el nuevo texto del token.
setFila
Descripción: Establece la fila en la que se encuentra el token.
Parámetros:
fila: Entero que representa la nueva fila del token.
setColumna
Descripción: Establece la columna en la que se encuentra el token.
Parámetros:
columna: Entero que representa la nueva columna del token.
setColor
Descripción: Establece el color asociado al token.
Parámetros:
color: Cadena de texto que representa el nuevo color del token.