MANUAL TECNICO
ANALIZADOR LEXICO

LENGUAJE: JAVA

JDK : 17

FRONTEND:

Clase Interface

Descripción General

La clase Interface extiende JFrame y proporciona una interfaz gráfica para el analizador léxico. Esta interfaz permite al usuario ingresar código fuente, visualizar los tokens generados y optimizar el código. Además, incluye funcionalidades para generar reportes de tokens en una nueva ventana.

Funcionalidad

Constructor Interface(): Configura la interfaz gráfica, incluyendo el look and feel, el tamaño de la ventana, y los componentes gráficos como JTextArea, JTable, y botones.
Método actualizarTabla(List<Token> tokensValidados): Actualiza la tabla con los tokens validados obtenidos del analizador léxico.

Componentes Utilizados

JTextArea textArea1: Área de texto editable donde el usuario ingresa el código fuente.
JTextArea textArea2: Nueva área de texto no editable que muestra el código optimizado.
JTable table: Tabla que muestra los tokens generados por el analizador léxico.
JScrollPane scrollPane: Contenedor con barra de desplazamiento para textArea1.
JScrollPane textArea2ScrollPane: Contenedor con barra de desplazamiento para textArea2.
JScrollPane tableScrollPane: Contenedor con barra de desplazamiento para la tabla de tokens.
JPanel jPanel1: Panel que contiene scrollPane.
JButton boton: Botón “Aceptar” que inicia el análisis del código fuente.
JButton saveButton: Botón “Generar Reporte HTML” (acción no implementada en el código proporcionado).
JMenuBar menuBar: Barra de menú que contiene el menú de reportes.
JMenu reportesMenu: Menú “Reportes” que contiene opciones de reportes.
JMenuItem reporteTokensItem: Opción de menú para generar un reporte de tokens.
Configuración de la Interfaz
Look and Feel: Se aplica el look and feel FlatLightLaf.

Layout: Se utiliza GridBagLayout para organizar los componentes en la ventana.
Acciones de los Componentes
Botón “Aceptar”: Al hacer clic, se obtiene el código fuente de textArea1, se analiza con analizador, se actualiza la tabla con los tokens validados y se optimiza el código mostrando el resultado en textArea2.
Menú “Reporte de tokens”: Al seleccionar esta opción, se abre una nueva ventana que muestra un reporte de los tokens en una tabla.

Clase LineNumberingTextArea

Descripción General

La clase LineNumberingTextArea extiende JTextArea y proporciona una funcionalidad adicional para mostrar números de línea junto a un JTextArea dado. Esta clase es útil para editores de texto o entornos de desarrollo donde es importante visualizar los números de línea.

Funcionalidad

Constructor LineNumberingTextArea(JTextArea textArea): Inicializa la instancia de LineNumberingTextArea y añade un DocumentListener al JTextArea proporcionado para actualizar los números de línea cada vez que el documento cambia.

Método updateLineNumbers(): Actualiza los números de línea en función del contenido actual del JTextArea asociado.

Componentes Utilizados

JTextArea textArea: El área de texto asociada a la cual se le añaden los números de línea.

Configuración de la Interfaz

DocumentListener: Se añade al documento del textArea para escuchar eventos de inserción, eliminación y cambio, y actualizar los números de línea en consecuencia.

Acciones de los Componentes

Método insertUpdate(DocumentEvent e): Llamado cuando se inserta texto en el textArea, actualiza los números de línea.

Método removeUpdate(DocumentEvent e): Llamado cuando se elimina texto del textArea, actualiza los números de línea.

Método changedUpdate(DocumentEvent e): Llamado cuando se cambia el texto del textArea, actualiza los números de línea.

BACKEND:

Clase Token

Descripción General

La clase Token representa un token individual en el proceso de análisis léxico. Un token es una secuencia de caracteres que se agrupan como una unidad significativa para el análisis sintáctico. Esta clase encapsula la información relevante de cada token, como su valor, expresión regular asociada, lenguaje, tipo, y su posición en el código fuente.

Funcionalidad

Constructor Token(String token, String expresionRegular, String lenguaje, String tipo, int fila, int columna): Inicializa un nuevo objeto Token con los valores proporcionados. Este constructor permite establecer todos los atributos del token al momento de su creación.
Métodos get y set: Proporcionan acceso y modificación a los atributos del token. Estos métodos permiten obtener y actualizar el valor del token, su expresión regular, lenguaje, tipo, y su posición en términos de fila y columna.

Atributos

String token: Representa el valor del token, es decir, la secuencia de caracteres que conforman el token.
String expresionRegular: La expresión regular que define el patrón del token.
String lenguaje: El lenguaje de programación al que pertenece el token.
String tipo: El tipo de token, que puede ser una palabra clave, un identificador, un operador, etc.
int fila: La fila en el código fuente donde se encuentra el token.
int columna: La columna en el código fuente donde se encuentra el token.

Métodos

public String getToken(): Devuelve el valor del token.
public void setToken(String token): Establece el valor del token.
public String getExpresionRegular(): Devuelve la expresión regular del token.
public void setExpresionRegular(String expresionRegular): Establece la expresión regular del token.
public String getLenguaje(): Devuelve el lenguaje del token.
public void setLenguaje(String lenguaje): Establece el lenguaje del token.
public String getTipo(): Devuelve el tipo del token.
public void setTipo(String tipo): Establece el tipo del token.
public int getFila(): Devuelve la fila del token.
public void setFila(int fila): Establece la fila del token.
public int getColumna(): Devuelve la columna del token.
public void setColumna(int columna): Establece la columna del token.
@Override public String toString(): Devuelve una representación en cadena del token, incluyendo todos sus atributos.
Explicación del Funcionamiento
La clase Token es fundamental para el análisis léxico, ya que cada instancia de Token representa una unidad léxica identificada en el código fuente. Al analizar el código, el analizador léxico crea objetos Token para cada secuencia de caracteres que coincide con una expresión regular definida. Estos tokens se utilizan posteriormente en el análisis sintáctico y semántico del código.

El constructor de la clase permite inicializar todos los atributos del token, asegurando que cada token tenga toda la información necesaria para su identificación y uso posterior. Los métodos get y set proporcionan una forma de acceder y modificar estos atributos, lo que es útil para ajustar y validar los tokens durante el proceso de análisis.

Clase AnalizadorLexico

Descripción General

La clase AnalizadorLexico es responsable de analizar el código fuente y generar tokens que representan las unidades léxicas del código. Este analizador puede manejar múltiples lenguajes (HTML, CSS, JavaScript) y cambiar dinámicamente entre ellos según el contenido del código.

Funcionalidad

Constructor AnalizadorLexico(): Inicializa las variables necesarias para el análisis léxico, incluyendo las posiciones actuales, listas de tokens y errores, y las instancias de los analizadores específicos para HTML, CSS y JavaScript.
Método analizarCodigo(String codigo): Analiza el código fuente línea por línea, generando tokens y validándolos según el lenguaje actual.
Método cambiarAnalizador(String linea): Cambia el analizador actual basado en una línea de comando específica en el código fuente.
Método tokenizarCSS(String linea): Tokeniza una línea de código CSS en sus componentes léxicos.

Atributos

String codigoFuente: El código fuente que se va a analizar.
int posicionActual: La posición actual en el código fuente.
int lineaActual: La línea actual en el código fuente.
int columnaActual: La columna actual en el código fuente.
List<Token> tokens: Lista de tokens generados durante el análisis.
List<String> errores: Lista de errores encontrados durante el análisis.
String analizadorActual: El analizador actual en uso (HTML, CSS, JavaScript).
List<Token> tokensValidados: Lista de tokens validados.
AnalizadorCSS analizadorCSS: Instancia del analizador CSS.
AnalizadorJS analizadorJS: Instancia del analizador JS.
AnalizadorHTML analizadorHTML: Instancia del analizador HTML.
TokenHTML tokenHTML: Instancia del tokenizador HTML.

Explicación del Funcionamiento

Constructor AnalizadorLexico(): Inicializa todas las variables necesarias para el análisis léxico. Esto incluye la posición actual en el código, la línea y columna actuales, y las listas para almacenar tokens y errores. También inicializa las instancias de los analizadores específicos para HTML, CSS y JavaScript.
Método analizarCodigo(String codigo): Este método es el núcleo del analizador léxico. Divide el código fuente en líneas y procesa cada línea individualmente. Si una línea está vacía, simplemente se salta. Si una línea contiene un comando para cambiar el analizador (por ejemplo, >>[html]), cambia el analizador actual. Luego, dependiendo del analizador actual, tokeniza la línea y valida los tokens generados.

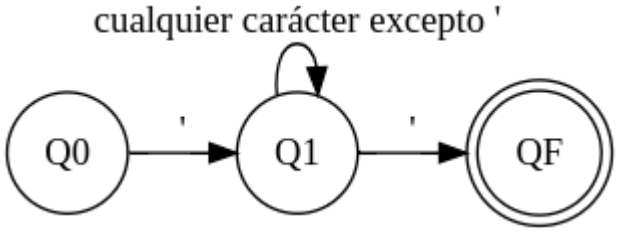
Método cambiarAnalizador(String linea): Este método cambia el analizador actual basado en una línea de comando específica. Por ejemplo, si la línea es >>[html], el analizador actual se cambia a HTML.
Método tokenizarCSS(String linea): Este método tokeniza una línea de código CSS. Divide la línea en tokens basados en caracteres específicos como {, }, :, ;, y espacios. Cada token se añade a la lista de tokens. lo mismo para cada uno de los lenguajes segun sea el caso

Clase AnalizadorCSS
Descripción General
La clase AnalizadorCSS se encarga de analizar y validar tokens específicos del lenguaje CSS. Utiliza varios autómatas para identificar y clasificar diferentes tipos de tokens CSS, como etiquetas, reglas, combinadores, colores, cadenas, enteros, identificadores, clases e IDs.

Funcionalidad
Constructor AnalizadorCSS(): Inicializa los autómatas necesarios para el análisis de tokens CSS.
Método analizarTokens(List<String> tokens, int fila): Analiza y valida una lista de tokens CSS, clasificándolos según su tipo y generando una lista de objetos Token con la información correspondiente.
Atributos
AutomataEtiquetaCSS automataEtiquetaCSS: Autómata para validar etiquetas o tipos CSS.
AutomataReglasCSS automataReglasCSS: Autómata para validar reglas CSS.
AutomataCombinadoresCSS automataCombinadoresCSS: Autómata para validar combinadores CSS.
AutomataColoresCSS automataColoresCSS: Autómata para validar colores CSS.
AutomataCadenasCSS automataCadenasCSS: Autómata para validar cadenas CSS.
AutomataEnterosCSS automataEnterosCSS: Autómata para validar enteros CSS.
AutomataOtrosCSS automataOtrosCSS: Autómata para validar otros elementos CSS.
AutomataIdentificadoresCSS automataIdentificadoresCSS: Autómata para validar identificadores CSS.
AutomataClaseCSS automataClaseCSS: Autómata para validar clases CSS.
AutomataIdCSS automataIdCSS: Autómata para validar IDs CSS.
Explicación del Funcionamiento
Constructor AnalizadorCSS(): Este constructor inicializa todos los autómatas necesarios para el análisis de tokens CSS. Cada autómata está especializado en reconocer y validar un tipo específico de token CSS.
Método analizarTokens(List<String> tokens, int fila): Este método recibe una lista de tokens y la fila en la que se encuentran. Para cada token, determina su tipo utilizando los autómatas correspondientes. Si un token es válido según un autómata, se clasifica como tal y se crea un objeto Token con la información del token, su tipo, y su posición en el código (fila y columna). Finalmente, se devuelve una lista de tokens validados.

Clase AutomataCadenasCSS
Descripción General
La clase AutomataCadenasCSS implementa un autómata finito determinista (DFA) para validar cadenas en CSS. Este autómata reconoce cadenas delimitadas por comillas simples (').

Funcionalidad
Constructor AutomataCadenasCSS(): Inicializa el autómata en su estado inicial (Q0).
Método esCadenaValida(String token): Valida si una cadena dada es una cadena CSS válida según las reglas del autómata.
Atributos
Estado estadoActual: Representa el estado actual del autómata.
Explicación del Funcionamiento
Constructor AutomataCadenasCSS(): Inicializa el estado del autómata a Q0, que es el estado inicial.
Método esCadenaValida(String token): Este método recorre cada carácter del token proporcionado y cambia el estado del autómata según las reglas definidas:
Estado Q0: Si encuentra una comilla simple ('), cambia al estado Q1. De lo contrario, la cadena no es válida.
Estado Q1: Permanece en Q1 mientras no encuentre otra comilla simple. Si encuentra una comilla simple, cambia al estado QF (estado de aceptación).
Estado QF: No debería haber más caracteres después de la comilla de cierre. Si hay más caracteres, la cadena no es válida.
El método devuelve true si el estado final del autómata es QF, indicando que la cadena es válida.



Clase AutomataIdentificadoresCSS

Descripción General

La clase AutomataIdentificadoresCSS implementa un autómata finito determinista (DFA) para validar identificadores en CSS. Un identificador en CSS puede comenzar con una letra y puede contener letras, dígitos y guiones (-).

Funcionalidad

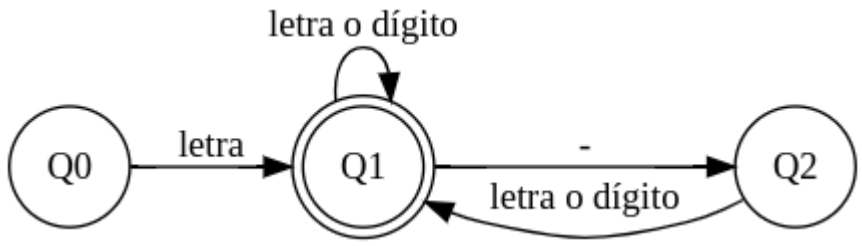
Constructor AutomataIdentificadoresCSS(): Inicializa el autómata en su estado inicial (Q0).
Método esIdentificadorValido(String token): Valida si una cadena dada es un identificador CSS válido según las reglas del autómata.

Atributos

Estado estadoActual: Representa el estado actual del autómata.

Explicación del Funcionamiento

Constructor AutomataIdentificadoresCSS(): Inicializa el estado del autómata a Q0, que es el estado inicial.
Método esIdentificadorValido(String token): Este método recorre cada carácter del token proporcionado y cambia el estado del autómata según las reglas definidas:
Estado Q0: Si encuentra una letra, cambia al estado Q1. De lo contrario, la cadena no es válida.
Estado Q1: Permanece en Q1 mientras encuentre letras o dígitos. Si encuentra un guion (-), cambia al estado Q2. Cualquier otro carácter invalida la cadena.
Estado Q2: Si encuentra una letra o un dígito, vuelve al estado Q1. Cualquier otro carácter invalida la cadena.
El método devuelve true si el estado final del autómata es Q1, indicando que el identificador es válido.



Clase AnalizadorJS

Descripción General

La clase AnalizadorJS se encarga de analizar y validar tokens específicos del lenguaje JavaScript. Utiliza varios autómatas para identificar y clasificar diferentes tipos de tokens JavaScript, como palabras reservadas, identificadores, booleanos, cadenas, decimales, enteros, operadores aritméticos, relacionales, símbolos, lógicos e incrementales.

Funcionalidad

Constructor AnalizadorJS(): Inicializa los autómatas necesarios para el análisis de tokens JavaScript.

Método analizarTokens(List<String> tokens, int fila): Analiza y valida una lista de tokens JavaScript, clasificándolos según su tipo y generando una lista de objetos Token con la información correspondiente.

Atributos

AutomataPalabrasReservadasJS automataPalabrasReservadasJS: Autómata para validar palabras reservadas de JavaScript.

AutomataIdentificadoresJS automataIdentificadoresJS: Autómata para validar identificadores de JavaScript.

AutomataBooleanJS automataBooleanJS: Autómata para validar valores booleanos de JavaScript.

AutomataCadenaJS automataCadenaJS: Autómata para validar cadenas de JavaScript.

AutomataDecimalesJS automataDecimalesJS: Autómata para validar números decimales de JavaScript.

AutomataEnterosJS automataEnterosJS: Autómata para validar números enteros de JavaScript.

AutomataAritmeticosJS automataAritmeticosJS: Autómata para validar operadores aritméticos de JavaScript.

AutomataRelacionalesJS automataRelacionalesJS: Autómata para validar operadores relacionales de JavaScript.

AutomataSimbolosJS automataSimbolosJS: Autómata para validar símbolos de JavaScript.

AutomataLogicoJS automataLogicoJS: Autómata para validar operadores lógicos de JavaScript.

AutomataIncrementalJS automataIncrementalJS: Autómata para validar operadores incrementales de JavaScript.

Explicación del Funcionamiento

Constructor AnalizadorJS(): Este constructor inicializa todos los autómatas necesarios para el análisis de tokens JavaScript. Cada autómata está especializado en reconocer y validar un tipo específico de token JavaScript.

Método analizarTokens(List<String> tokens, int fila): Este método recibe una lista de tokens y la fila en la que se encuentran. Para cada token, determina su tipo utilizando los autómatas correspondientes. Si un token es válido según un autómata, se clasifica como tal y se crea un objeto Token con la información del token, su tipo, y su posición en el código (fila y columna). Finalmente, se devuelve una lista de tokens validados.

Clase AutomataIdentificadoresJS

Descripción General

La clase AutomataIdentificadoresJS implementa un autómata finito determinista (DFA) para validar identificadores en JavaScript. Un identificador en JavaScript puede comenzar con una letra y puede contener letras, dígitos y guiones (-).

Funcionalidad

Constructor AutomataIdentificadoresJS(): Inicializa el autómata en su estado inicial (Q0).
Método esIdentificadorValido(String token): Valida si una cadena dada es un identificador JavaScript válido según las reglas del autómata.

Atributos

Estado estadoActual: Representa el estado actual del autómata.

Explicación del Funcionamiento

Constructor AutomataIdentificadoresJS(): Inicializa el estado del autómata a Q0, que es el estado inicial.

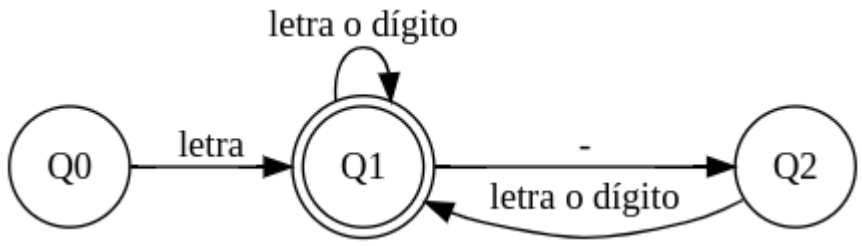
Método esIdentificadorValido(String token): Este método recorre cada carácter del token proporcionado y cambia el estado del autómata según las reglas definidas:

Estado Q0: Si encuentra una letra, cambia al estado Q1. De lo contrario, la cadena no es válida.

Estado Q1: Permanece en Q1 mientras encuentre letras o dígitos. Si encuentra un guion (-), cambia al estado Q2. Cualquier otro carácter invalida la cadena.

Estado Q2: Si encuentra una letra o un dígito, vuelve al estado Q1. Cualquier otro carácter invalida la cadena.

El método devuelve true si el estado final del autómata es Q1, indicando que el identificador es válido.



Clase AutomataPalabrasReservadasJS

Descripción General

La clase AutomataPalabrasReservadasJS implementa un autómata finito determinista (DFA) para validar palabras reservadas en JavaScript. Este autómata utiliza un conjunto de palabras reservadas predefinidas y verifica si un token pertenece a este conjunto.

Funcionalidad

Constructor AutomataPalabrasReservadasJS(): Inicializa el autómata y el conjunto de palabras reservadas.

Método esPalabraReservada(String token): Verifica si un token es una palabra reservada.

Método procesarToken(String token, int fila, int columna): Procesa un token para determinar si es una palabra reservada válida y devuelve un objeto Token si lo es.

Atributos

Set<String> palabrasReservadas: Conjunto de palabras reservadas en JavaScript.

Estado estadoActual: Representa el estado actual del autómata.

Explicación del Funcionamiento

Constructor AutomataPalabrasReservadasJS(): Inicializa el conjunto de palabras reservadas y establece el estado inicial del autómata a Q0.

Método inicializarPalabrasReservadas(): Añade todas las palabras reservadas de JavaScript al conjunto palabrasReservadas.

Método esPalabraReservada(String token): Comprueba si el token proporcionado está en el conjunto de palabras reservadas.

Método procesarToken(String token, int fila, int columna): Este método recorre cada carácter del token y cambia el estado del autómata según las reglas definidas:

Estado Q0: Si encuentra una letra, cambia al estado Q1. De lo contrario, cambia al estado ERROR.

Estado Q1: Permanece en Q1 mientras encuentre letras o dígitos. Cualquier otro carácter cambia al estado ERROR.

Estado ERROR: Indica que el token no es válido.

Si el estado final del autómata es Q1 y el token es una palabra reservada, el método devuelve un objeto Token con la información correspondiente.

Clase AutomataRelacionalesJS

Descripción General

La clase AutomataRelacionalesJS implementa un autómata finito determinista (DFA) para validar operadores relacionales en JavaScript. Este autómata reconoce operadores como ==, !=, <=, >=, <, y >.

Funcionalidad

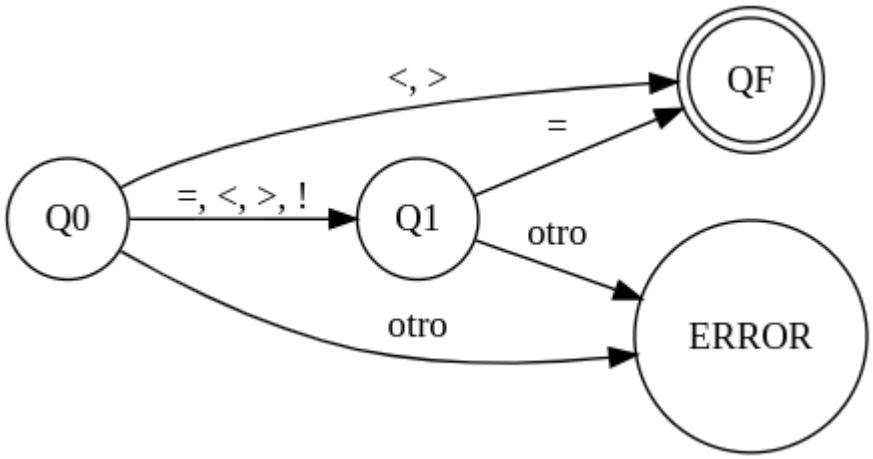
Constructor AutomataRelacionalesJS(): Inicializa el autómata en su estado inicial (Q0).
Método esRelacionalValido(String token): Valida si una cadena dada es un operador relacional JavaScript válido según las reglas del autómata.
Método unirYValidarTokens(List<String> tokens): Une una lista de tokens y valida si la cadena resultante es un operador relacional válido.

Atributos

Estado estadoActual: Representa el estado actual del autómata.

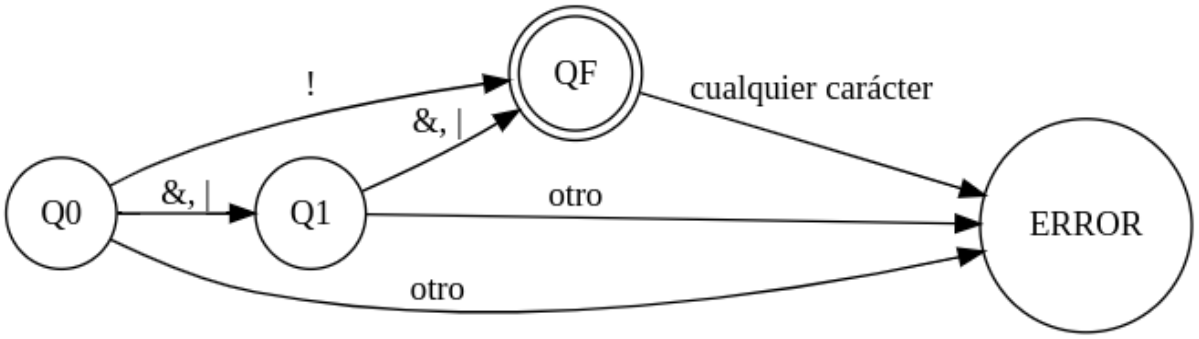
Explicación del Funcionamiento

Constructor AutomataRelacionalesJS(): Inicializa el estado del autómata a Q0, que es el estado inicial.
Método esRelacionalValido(String token): Este método recorre cada carácter del token proporcionado y cambia el estado del autómata según las reglas definidas:
Estado Q0: Si encuentra =, <, >, o !, cambia al estado Q1. Cualquier otro carácter cambia al estado ERROR.
Estado Q1: Si encuentra =, cambia al estado QF. Cualquier otro carácter cambia al estado ERROR.
Estado QF: Estado de aceptación.
Estado ERROR: Indica que el token no es válido.
El método también valida operadores de un solo carácter (<, >).
Método unirYValidarTokens(List<String> tokens): Une una lista de tokens en una sola cadena y valida si es un operador relacional válido utilizando el método esRelacionalValido.



Clase AutomataLogicoJS
Descripción General
La clase AutomataLogicoJS implementa un autómata finito determinista (DFA) para validar operadores lógicos en JavaScript. Este autómata reconoce operadores como !, &&, y ||.

Funcionalidad
Constructor AutomataLogicoJS(): Inicializa el autómata en su estado inicial (Q0).
Método esLogicoValido(String token): Valida si una cadena dada es un operador lógico JavaScript válido según las reglas del autómata.
Atributos
Estado estadoActual: Representa el estado actual del autómata.
Explicación del Funcionamiento
Constructor AutomataLogicoJS(): Inicializa el estado del autómata a Q0, que es el estado inicial.
Método esLogicoValido(String token): Este método recorre cada carácter del token proporcionado y cambia el estado del autómata según las reglas definidas:
Estado Q0: Si encuentra !, cambia al estado QF. Si encuentra & o |, cambia al estado Q1. Cualquier otro carácter cambia al estado ERROR.
Estado Q1: Si encuentra & o |, cambia al estado QF. Cualquier otro carácter cambia al estado ERROR.
Estado QF: Estado de aceptación. No debería haber más caracteres después del operador.
Estado ERROR: Indica que el token no es válido.
El método devuelve true si el estado final del autómata es QF, indicando que el operador lógico es válido.



Clase AutomataSimbolosJS

Descripción General

La clase AutomataSimbolosJS implementa un autómata finito determinista (DFA) para validar símbolos en JavaScript. Este autómata reconoce símbolos como paréntesis, corchetes, llaves, signos de asignación, punto y coma, coma, punto y dos puntos.

Funcionalidad

Constructor AutomataSimbolosJS(): Inicializa el autómata en su estado inicial (Q0) y configura el conjunto de símbolos válidos.

Método esSimboloValido(String token): Valida si una cadena dada es un símbolo JavaScript válido según las reglas del autómata.

Método obtenerTipoSimbolo(String token): Devuelve el tipo de símbolo si el token es válido.

Atributos

Estado estadoActual: Representa el estado actual del autómata.

Map<Character, String> simbolosValidos: Mapa que contiene los símbolos válidos y sus descripciones.

Explicación del Funcionamiento

Constructor AutomataSimbolosJS(): Inicializa el estado del autómata a Q0 y configura el mapa de símbolos válidos mediante el método inicializarSimbolosValidos().

Método inicializarSimbolosValidos(): Añade todos los símbolos válidos de JavaScript al mapa simbolosValidos.

Método esSimboloValido(String token): Este método recorre cada carácter del token proporcionado y cambia el estado del autómata según las reglas definidas:

Estado Q0: Si encuentra un símbolo válido, cambia al estado QF. Cualquier otro carácter cambia al estado ERROR.

Estado QF: Estado de aceptación. No debería haber más caracteres después del símbolo.

Estado ERROR: Indica que el token no es válido.

El método devuelve true si el estado final del autómata es QF, indicando que el símbolo es válido.

Método obtenerTipoSimbolo(String token): Si el token es válido, devuelve la descripción del símbolo desde el mapa simbolosValidos.