DIVISIÓN DE CIENCIAS DE LA INGENIERÍA
LENGUAJES FORMALES DE PROGRAMACIÓN.

Manual Técnico - "Analizador Léxico"

El IDE utilizado fue: NetBeans 20.

Con la versión de java: 17.0.6.

1. Clase AnalizadorLexico

La clase AnalizadorLexico es responsable de realizar el análisis léxico del código fuente proporcionado. Identifica tokens, los clasifica según su tipo, y los pinta en una cuadrícula visualizada en un lienzo. A continuación, se describe cada atributo y método de la clase:

Métodos:

1. **AnalizadorLexico(String codigoFuente, Cuadricula cuadricula, LienzoPanel lienzoPanel)**:
   - **Descripción**: Constructor que inicializa los atributos de la clase.
   - **Parámetros**:
     - codigoFuente: Código fuente que se analizará.
     - cuadricula: Referencia a la cuadrícula donde se pintarán los tokens.
     - lienzoPanel: Panel de lienzo donde se visualizarán los tokens pintados.
   - **Funcionalidad**: Inicializa el arreglo de tokens, el contador de tokens, y asocia el código fuente, la cuadrícula y el panel de lienzo.
2. **analizar()**:
   - **Descripción**: Método principal que analiza el código fuente línea por línea y palabra por palabra.
   - **Funcionalidad**: Recorre el código fuente, identifica tokens especiales y comunes, y los procesa utilizando el método procesarLinea. También maneja la secuencia de filas y columnas para la cuadrícula.
3. **procesarLinea(String linea, int numeroLinea, int secuenciaFila, int secuenciaColumna)**:
   - **Descripción**: Procesa una línea completa del código fuente.
   - **Parámetros**:
     - linea: Línea de código a procesar.
     - numeroLinea: Número de la línea en el código fuente.
     - secuenciaFila: Número de la fila en la cuadrícula para el token.
     - secuenciaColumna: Número de la columna en la cuadrícula para el token.
   - **Funcionalidad**: Identifica tokens especiales como Square.Color y los pinta en la cuadrícula. Si el token especial contiene fila y columna, se utiliza esa posición; de lo contrario, se asigna de manera secuencial.
4. **ProcesarLineaComun(String linea, int numeroLinea)**:
   - **Descripción**: Procesa el contenido de la línea que no contiene tokens especiales.
   - **Parámetros**:
     - linea: Línea de código a procesar.
     - numeroLinea: Número de la línea en el código fuente.

- **Funcionalidad**: Identifica diferentes tipos de tokens, como comentarios, palabras reservadas, booleanos, operadores lógicos, identificadores, enteros, decimales, caracteres, cadenas, entre otros. Cada tipo de token recibe un color específico y se almacena en el arreglo de tokens.

5. **agregarToken(Token token)**:
   - **Descripción**: Agrega un token al arreglo de tokens.
   - **Parámetros**:
     - token: Objeto Token a agregar.
   - **Funcionalidad**: Verifica que el arreglo de tokens no esté lleno, y si no lo está, almacena el nuevo token.

Métodos de validación para identificar tokens:

6. **EsEntero(String palabra)**:
   - **Descripción**: Verifica si una palabra es un número entero.
   - **Funcionalidad**: Intenta convertir la palabra en un entero usando Integer.parseInt. Si tiene éxito, la palabra es un entero; de lo contrario, no lo es.
7. **EsDecimal(String palabra)**:
   - **Descripción**: Verifica si una palabra es un número decimal.
   - **Funcionalidad**: Intenta convertir la palabra en un número decimal usando Double.parseDouble. Verifica que la palabra contenga un punto.
8. **EsBooleano(String palabra)**:
   - **Descripción**: Verifica si una palabra es un valor booleano.
   - **Funcionalidad**: Compara la palabra con los valores booleanos válidos (True o False).
9. **EsCaracter(String palabra)**:
   - **Descripción**: Verifica si una palabra es un carácter.
   - **Funcionalidad**: Verifica que la palabra tenga exactamente 3 caracteres y que comience y termine con acento grave (´).
10. **EsCadena(String palabra)**:
    - **Descripción**: Verifica si una palabra es una cadena.
    - **Funcionalidad**: Verifica que la palabra comience y termine con comillas dobles (" ").
11. **EsPalabraReservada(String palabra)**:
    - **Descripción**: Verifica si una palabra es una palabra reservada.
    - **Funcionalidad**: Compara la palabra con una lista de palabras reservadas.
12. **EsIdentificador(String palabra)**:
    - **Descripción**: Verifica si una palabra es un identificador.
    - **Funcionalidad**: Verifica si la palabra comienza con una letra y solo contiene letras, números y guiones bajos.
13. **EsAritmeticos(String palabra)**:
    - **Descripción**: Verifica si una palabra es un operador aritmético.
    - **Funcionalidad**: Compara la palabra con una lista de operadores aritméticos (+, -, *, etc.).

- **Autómata para Identificadores**

Para identificar nombres de variables o funciones que comienzan con una letra y pueden contener letras, dígitos y guiones bajos:

estado_inicial -> q1

q1 -> [a-zA-Z] -> q2

q2 -> [a-zA-Z0-9_] -> q2

q1: Estado inicial, transita a **q2** si el primer carácter es una letra.

q2: Estado que acepta letras, dígitos y guiones bajos, manteniéndose en **q2** mientras el carácter sea válido.

- **Autómata para Literales Enteros**

Para reconocer números enteros (secuencias de dígitos):

estado_inicial -> q1

q1 -> [0-9] -> q2

q2 -> [0-9] -> q2

q1: Estado inicial, transita a **q2** si el carácter es un dígito.

q2: Estado que acepta dígitos, manteniéndose en **q2** mientras el carácter sea un dígito.

- **Autómata para Literales Decimales**

Para números decimales que contienen un punto y dígitos antes y después del punto:

estado_inicial -> q1

q1 -> [0-9] -> q2

q2 -> . -> q3

q3 -> [0-9] -> q4

q4 -> [0-9] -> q4

q1: Estado inicial, transita a **q2** si el carácter es un dígito.

q2: Estado que acepta el punto decimal.

q3: Estado que acepta dígitos después del punto, manteniéndose en **q4** mientras el carácter sea un dígito.

- **Autómata para Cadenas de Texto**

Para identificar cadenas de texto entre comillas:

estado_inicial -> q1

q1 -> " -> q2

q2 -> [^"] -> q2

q2 -> " -> q3

q1: Estado inicial, transita a **q2** si el carácter es una comilla.

q2: Estado que acepta cualquier carácter excepto la comilla, manteniéndose en **q2** hasta encontrar otra comilla.

q3: Estado final que acepta la cadena entre comillas.

- **Autómata para Palabras Reservadas**

Para reconocer palabras reservadas específicas del lenguaje (como if, else, while):

Inicio -> q1 (lee 'i')

q1 -> q2 (lee 'f') -> q3 (fin, palabra "if")

Inicio -> q4 (lee 'e')

q4 -> q5 (lee 'l')

q5 -> q6 (lee 's')

q6 -> q7 (lee 'e') -> q8 (fin, palabra "else")

Inicio -> q9 (lee 'w')

q9 -> q10 (lee 'h')

q10 -> q11 (lee 'i')

q11 -> q12 (lee 'l')

q12 -> q13 (lee 'e') -> q14 (fin, palabra "while")

q2: Transita a **q3** si el siguiente carácter es f.

q3: Estado final si el siguiente carácter no es una letra, indicando fin de la palabra if.

1. **Clase Automata:**

La clase Automata se encarga de generar un autómata gráfico basado en el lexema de un Token utilizando Graphviz. El autómata representa la transición de estados a medida que se procesan los caracteres del lexema. Una vez generado, se muestra la imagen del autómata en una ventana gráfica utilizando Swing en Java.

Funcionalidad:

1. **Generación del archivo .dot:**
   Se crea un archivo de texto con extensión .dot que contiene las definiciones de los estados y transiciones del autómata.
2. **Uso de Graphviz:**
   Se ejecuta el comando dot de Graphviz para convertir el archivo .dot en una imagen .png.
3. **Manejo de Errores:**
   Si el proceso de generación de la imagen falla, se captura la excepción y se muestra un mensaje de error en la consola.

Método graficarAutomata()

Este método se encarga de generar un archivo .dot que describe el autómata basado en el lexema del Token, luego usa Graphviz para convertir ese archivo en una imagen PNG.

Funcionalidad:

1. **Generación del archivo .dot:**
   Se crea un archivo de texto con extensión .dot que contiene las definiciones de los estados y transiciones del autómata.
2. **Uso de Graphviz:**
   Se ejecuta el comando dot de Graphviz para convertir el archivo .dot en una imagen .png.
3. **Manejo de Errores:**
   Si el proceso de generación de la imagen falla, se captura la excepción y se muestra un mensaje de error en la consola.

Metodos:

- **Método mostrarInformacion()**

Este método combina la generación y la visualización del autómata. Primero llama a graficarAutomata() para crear la imagen, y luego llama a mostrarImagenAutomata() para mostrarla en una ventana.

- **Método mostrarImagenAutomata()**

Este método se encarga de cargar la imagen generada por Graphviz y mostrarla en un JFrame.

Funcionalidad:

1. **Verificación de la imagen:**
   Verifica que el archivo automata.png exista antes de intentar cargarlo.
2. **Carga y visualización:**
   Utiliza ImageIO.read() para cargar la imagen y JLabel para mostrarla dentro de un JFrame.
3. **Manejo de Errores:**
   Si la imagen no puede ser cargada, se muestra un mensaje de error.

4. Clase Cuadricula:

La clase Cuadricula es responsable de manejar una matriz bidimensional de tokens en una cuadrícula. Se utiliza para visualizar y gestionar el posicionamiento de tokens en una interfaz gráfica. Los tokens pueden tener colores asociados, y la cuadrícula permite asignar, dibujar, y actualizar estos tokens de forma secuencial o individual.

Métodos

- **Constructor**
  - Cuadricula(int filas, int columnas): Inicializa la cuadrícula con el número de filas y columnas especificadas. Crea la matriz de tokens y marca las áreas como no modificadas.
- **Acceso a Filas y Columnas**
  - int getFilas(): Devuelve el número de filas de la cuadrícula.
  - int getColumnas(): Devuelve el número de columnas de la cuadrícula.
- **Asignación de Tokens**
  - void setToken(int fila, int columna, Token token): Asigna un Token a una posición específica de la cuadrícula. La posición es validada para que esté dentro de los límites.
  - Token getToken(int fila, int columna): Obtiene el token en una posición específica. Devuelve null si la posición está fuera de los límites.
- **Dibujo de la Cuadrícula**
  - void dibujarCuadricula(Graphics g, int anchoCuadro, int altoCuadro): Dibuja la cuadrícula y sus tokens en el componente gráfico, rellenando los cuadros con el color del token correspondiente o en blanco si no hay token.
- **Asignación Secuencial de Tokens**

  - void asignarTokens(Token[] listaTokens, int tokenCount): Asigna tokens a la cuadrícula de manera secuencial, respetando el número máximo de tokens disponibles y la capacidad total de la cuadrícula.

- **Actualización de Color**
  - void cambiarColorCuadro(int fila, int columna, String colorHex): Actualiza el color de un cuadro en una posición específica. Si no existe un token en esa posición, crea uno con el color y posición asignada.
- **Detección de Áreas Modificadas**
  - boolean[][] getAreasModificadas(): Devuelve la matriz que indica cuáles áreas de la cuadrícula han sido modificadas.

5. Clase EditorTexto:

La clase EditorTexto extiende JTextArea y proporciona un editor de texto con características adicionales, como la capacidad de mostrar la posición actual del cursor (línea y columna) y la funcionalidad para cargar el contenido de archivos de texto desde el sistema de archivos. Esta clase está diseñada para integrarse en interfaces gráficas de usuario (GUI) construidas con Swing.

La clase EditorTexto extiende JTextArea y proporciona un editor de texto con características adicionales, como la capacidad de mostrar la posición actual del cursor (línea y columna) y la funcionalidad para cargar el contenido de archivos de texto desde el sistema de archivos. Esta clase está diseñada para integrarse en interfaces gráficas de usuario (GUI) construidas con Swing.

- **Métodos Principales:**
  - ✓ cargarArchivo(): Permite al usuario seleccionar un archivo desde su sistema mediante un diálogo de archivos y carga su contenido en el área de texto.
  - ✓ crearMenu(EditorTexto editor): Método estático que construye y devuelve una barra de menús con una opción para cargar archivos. Este menú está diseñado para integrarse fácilmente con la interfaz gráfica del editor.

6. Clase LienzoPanel:

La clase LienzoPanel es un componente personalizado de JPanel que se utiliza para mostrar visualmente una cuadrícula de tokens, donde cada cuadro está pintado con el color correspondiente al token que representa. Además, la clase permite la interacción con los cuadros de la cuadrícula, de modo que al hacer clic en un cuadro se muestra información detallada sobre el token y se genera su autómata. También incluye funcionalidad para guardar la cuadrícula como imagen en formatos PNG y JPG.

Metodos:

1. **Métodos Sobrescritos:**

- paintComponent(Graphics g): Método que dibuja la cuadrícula y los tokens. Cada cuadro se pinta según el color del token que contiene, y si no hay token, se pinta en blanco.
- Este método se asegura de que la cuadrícula se ajuste al tamaño disponible del panel.

2. **Métodos de Guardado de Imagen:**

- guardarImagenPNG(String rutaArchivo) y guardarImagenJPG(String rutaArchivo): Guardan la representación actual de la cuadrícula en una imagen en el formato especificado (PNG o JPG). Utilizan el método privado GuardarImagen que renderiza la cuadrícula en una BufferedImage y luego guarda la imagen en un archivo.

3. **Interacción con el Usuario:**

- Al hacer clic en cualquier cuadro de la cuadrícula, el MouseListener obtiene la fila y columna, recupera el token correspondiente y muestra su información (tipo, lexema, línea, columna, posición en la cuadrícula) en una ventana emergente (JOptionPane).
- Además, se instancia un objeto Automata que genera el autómata correspondiente al token seleccionado y muestra su información.

4. **Método setCuadricula:**

- Este método permite actualizar la cuadrícula que se va a visualizar en el panel, ajustando dinámicamente el tamaño del panel de acuerdo con el número de filas y columnas de la cuadrícula.

7. Clase Token:

La clase Token representa una entidad léxica identificada durante el análisis léxico de un código fuente. Cada token tiene atributos relacionados con su tipo, lexema, ubicación en el código (línea y columna), color asociado, y su posible ubicación en una cuadrícula gráfica. Esta clase permite crear, acceder y modificar tokens, así como gestionar su representación visual mediante colores.

Métodos:

- **Métodos getter:**
  - getTipo(): Retorna el tipo de token.
  - getLexema(): Retorna el texto del token.
  - getLinea(): Retorna la línea donde se encuentra el token.
  - getColumna(): Retorna la columna donde se encuentra el token.

  - getColor(): Retorna el color del token como un objeto Color.
  - getFilaCuadricula(): Retorna la fila del token en la cuadrícula, o -1 si no se especifica.
  - getColumnaCuadricula(): Retorna la columna del token en la cuadrícula, o -1 si no se especifica.
- **Método setter:**
  - setColor(String colorHex): Permite cambiar el color del token proporcionando un nuevo valor en formato hexadecimal.
- **Validación del formato hexadecimal del color:**
  - asegurarFormatoHexadecimal(String colorHex): Asegura que el formato del código hexadecimal es válido (empieza con # y tiene una longitud de 7 caracteres). Si el formato es incorrecto, lanza una excepción IllegalArgumentException.
- **Representación en texto:**
  - toString(): Devuelve una cadena con la información completa del token, incluyendo su tipo, lexema, ubicación en el código, color y coordenadas en la cuadrícula.