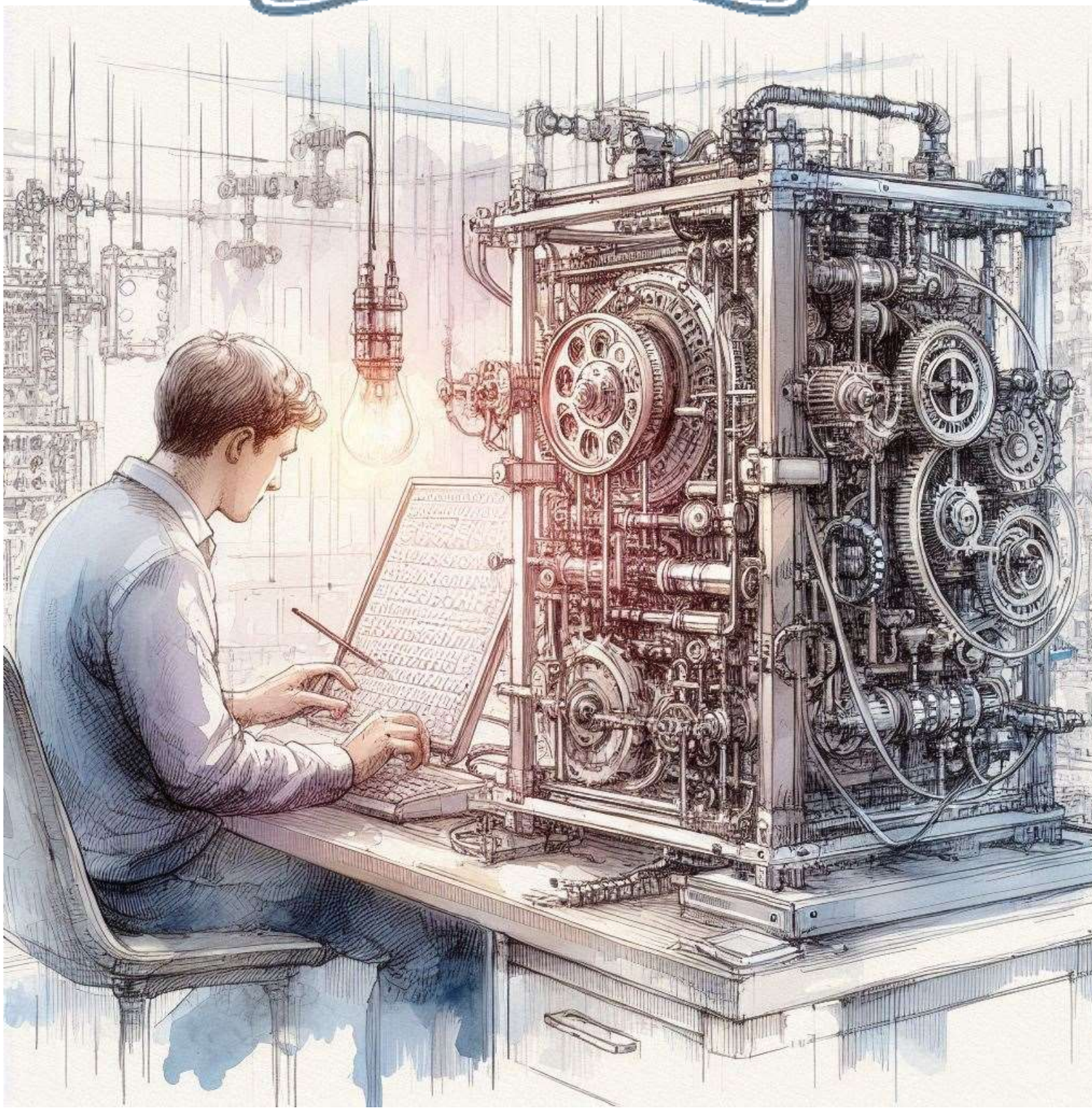
MANUAL DE USUARIO



Contenido

Analizadores Léxicos

Introducción

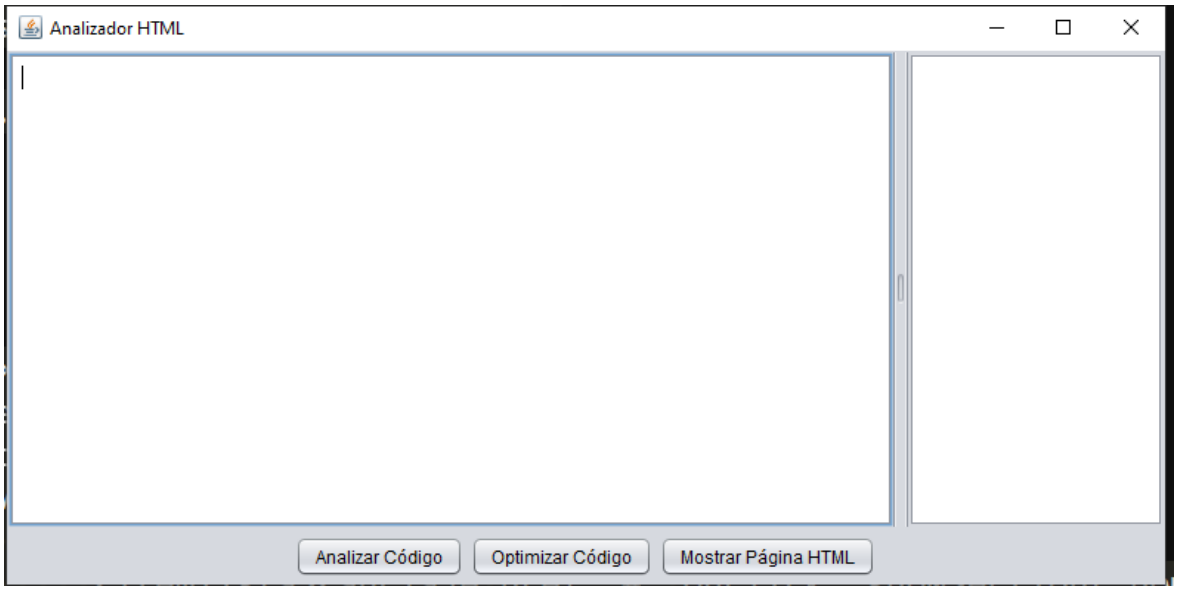
Este conjunto de clases incluye tres analizadores léxicos: `Analizadorhtml`, `AnalizadorCSS` y `AnalizadorJS`, que permiten analizar código fuente de HTML, CSS y JavaScript, respectivamente. Además, se incluye la clase `Token`, que se utiliza para representar las unidades léxicas extraídas durante el análisis. Cada analizador está diseñado para identificar y categorizar diferentes tipos de tokens dentro de su respectivo lenguaje.

Requisitos Previos

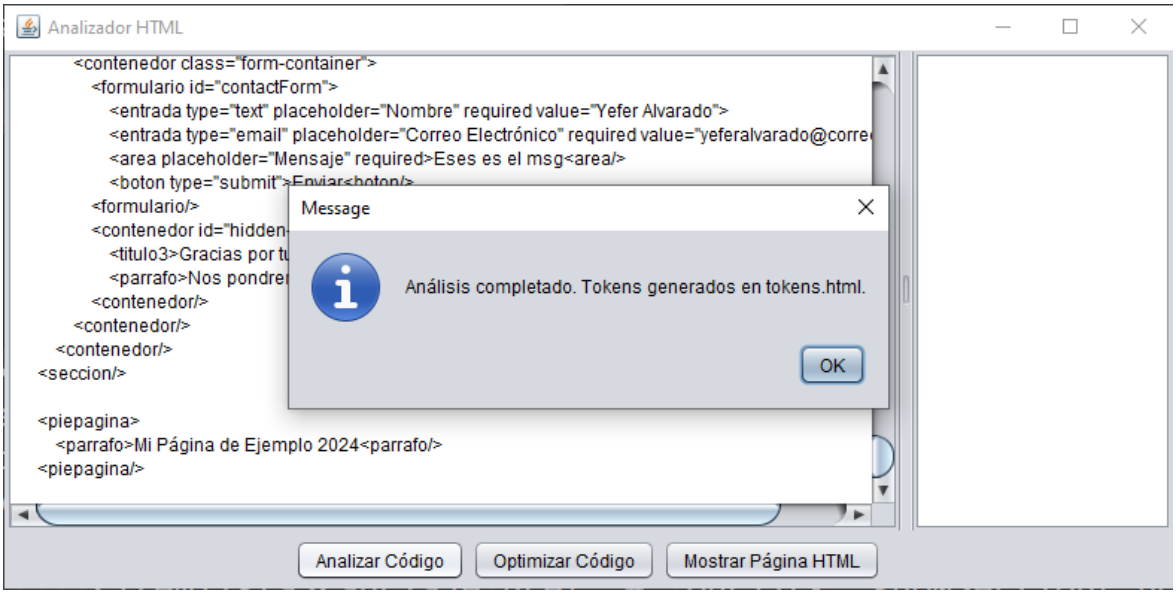
Antes de usar estas clases, asegúrate de tener:

- Un entorno de desarrollo Java configurado (JDK 17 o superior recomendado).
- Conocimientos básicos de programación en Java.

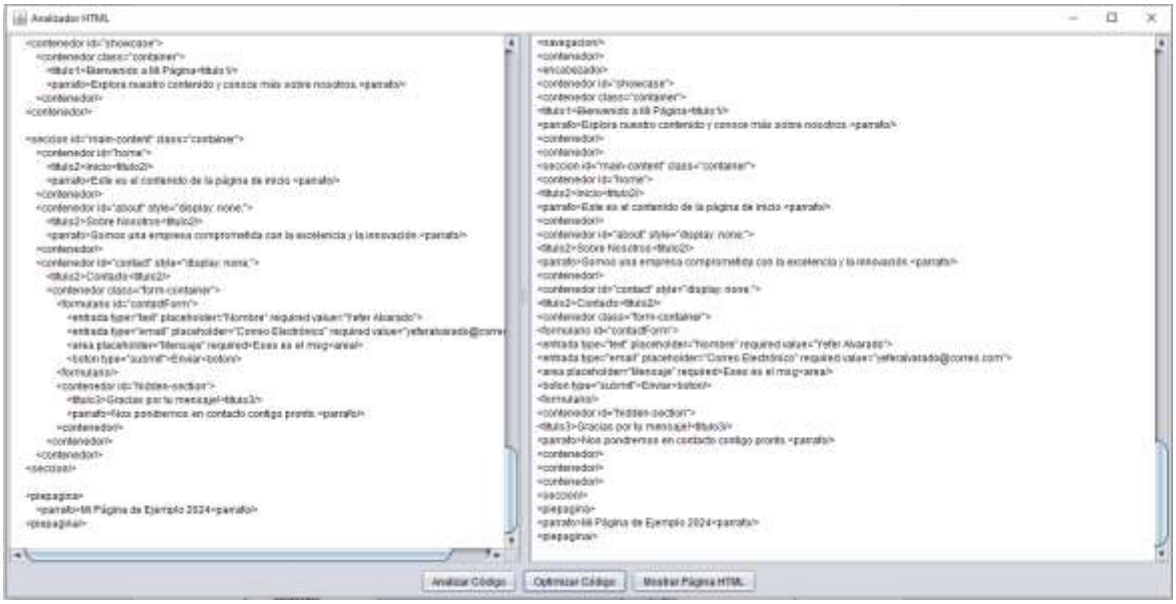
Al iniciar el programa se muestra la siguiente pantalla que contiene tres botones y una zona para ingresar el código



Al ingresar el código y oprimir el botón de analizar código se crearan los tokens

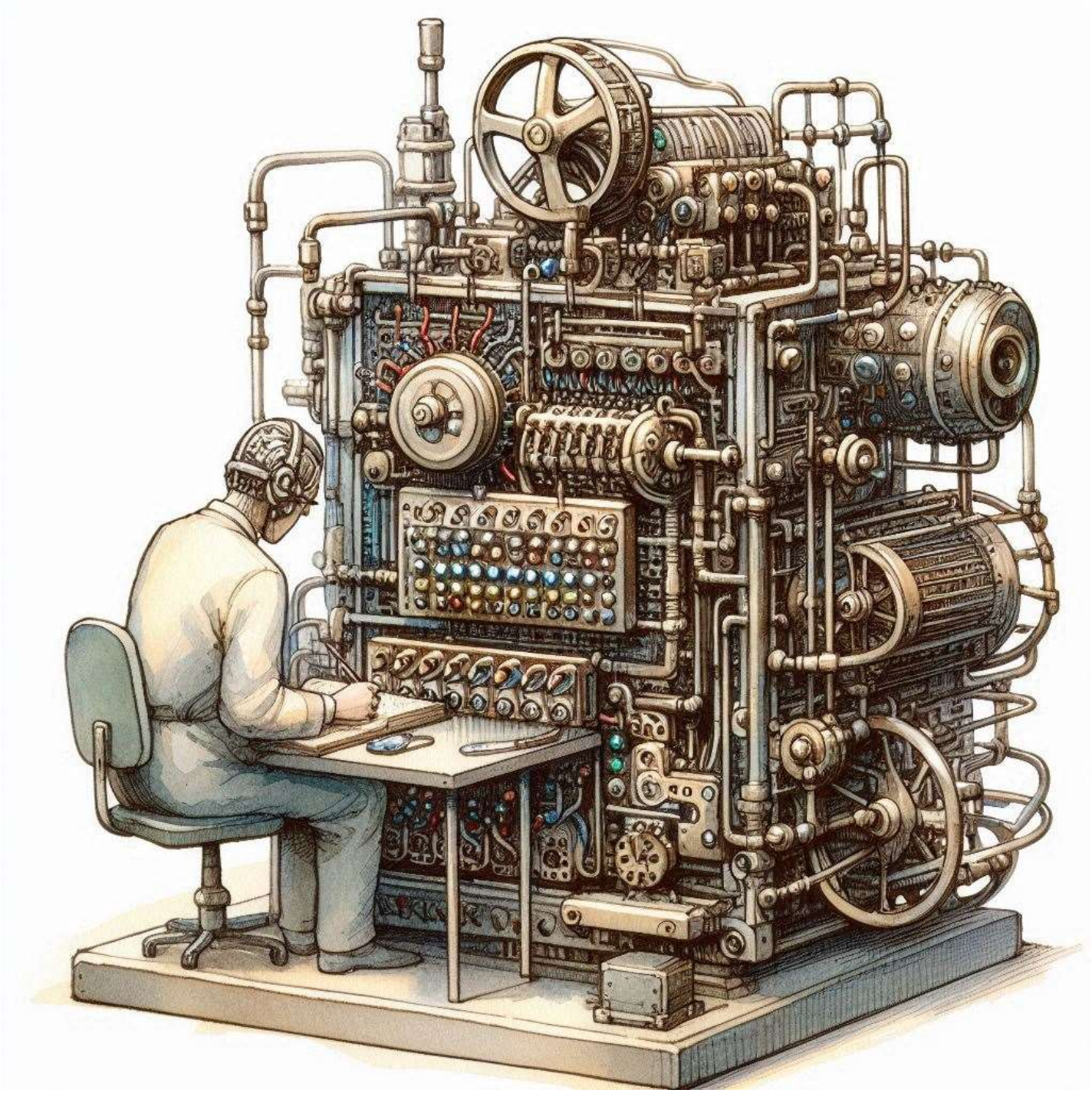


En el botón de optimizar código se eliminarán los espacios entre líneas y se borran los comentarios

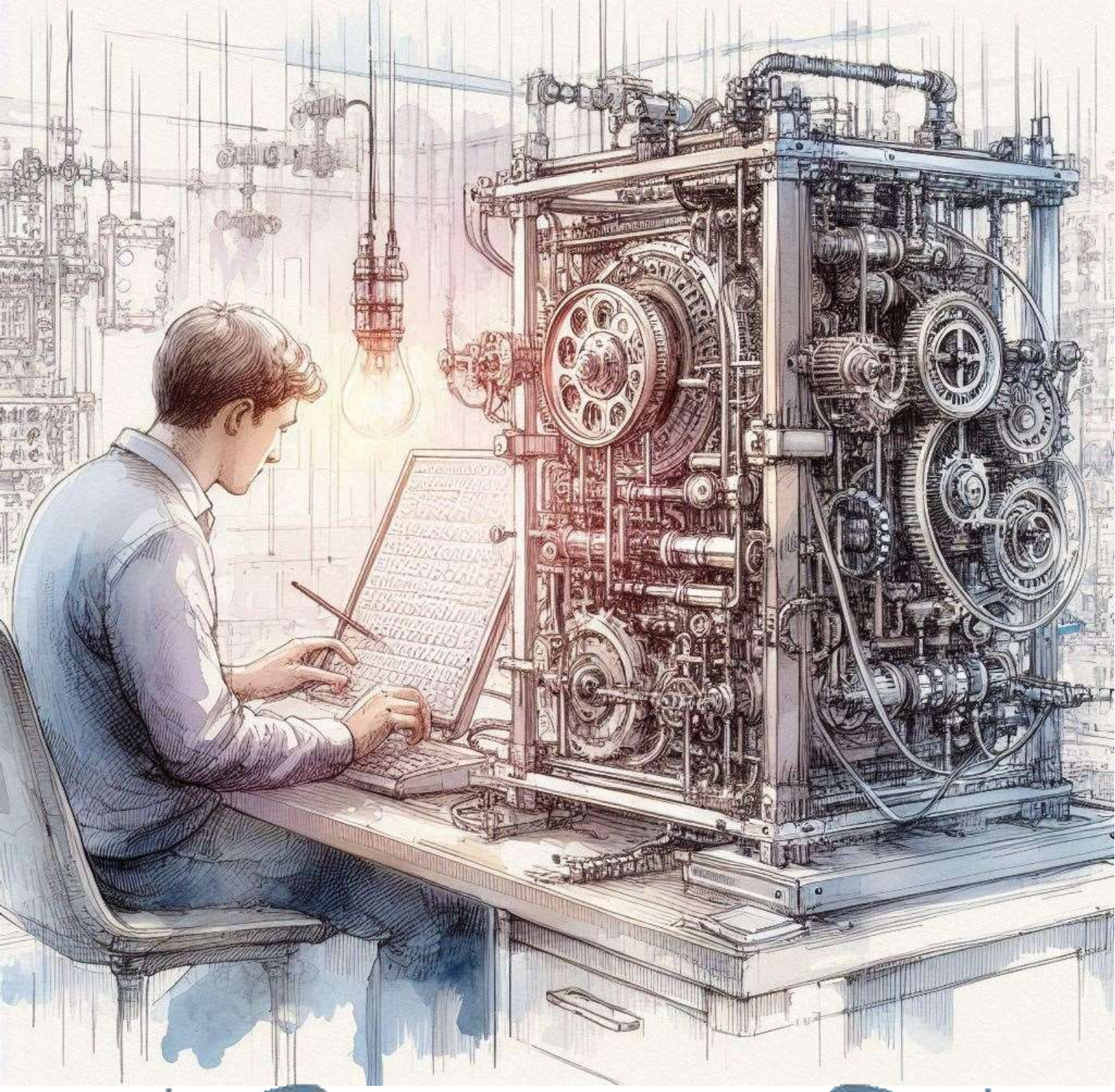


Y en el último botón se mostrara la página generada.

AUTOR



NOMBRE: VICTOR VEGA
CARNÉ: 202230772



MANUAL TÉCNICO

Contenido

HERRAMIENTAS UTILIZADAS

LENGUAJE DE PROGRAMACIÓN JAVA-VERSIÓN 17

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. El lenguaje Java proporciona:

- El paradigma de la programación orientada a objetos.
- Ejecución de un mismo programa en múltiples sistemas operativos y plataformas.
- Es posible utilizarlo para múltiples propósitos, desde aplicaciones de escritorio hasta en servidores web.
- Tiene una curva de aprendizaje media pero también toma lo mejor de otros lenguajes orientados a objetos, como C++.

IDE: SUBLIME TEXT

Sublime Text es un editor de texto avanzado que se utiliza principalmente para la programación y la codificación. Fue desarrollado por Jon Skinner en 2008 y se ha convertido en una herramienta popular entre los desarrolladores debido a su facilidad de uso y su gran variedad de funciones y características.

- Resaltado de sintaxis: Sublime Text es capaz de reconocer diferentes lenguajes de programación y resaltar la sintaxis de manera que el código sea más fácil de leer.
- Autocompletado: Sublime Text puede autocompletar el código en función de lo que el usuario está escribiendo, lo que acelera el proceso de escritura y reduce los errores. Búsqueda y reemplazo: Sublime Text tiene una herramienta de búsqueda y reemplazo avanzada que permite al usuario buscar y reemplazar cadenas de texto en todo el documento.

IDE: APACHE NETBEANS 17

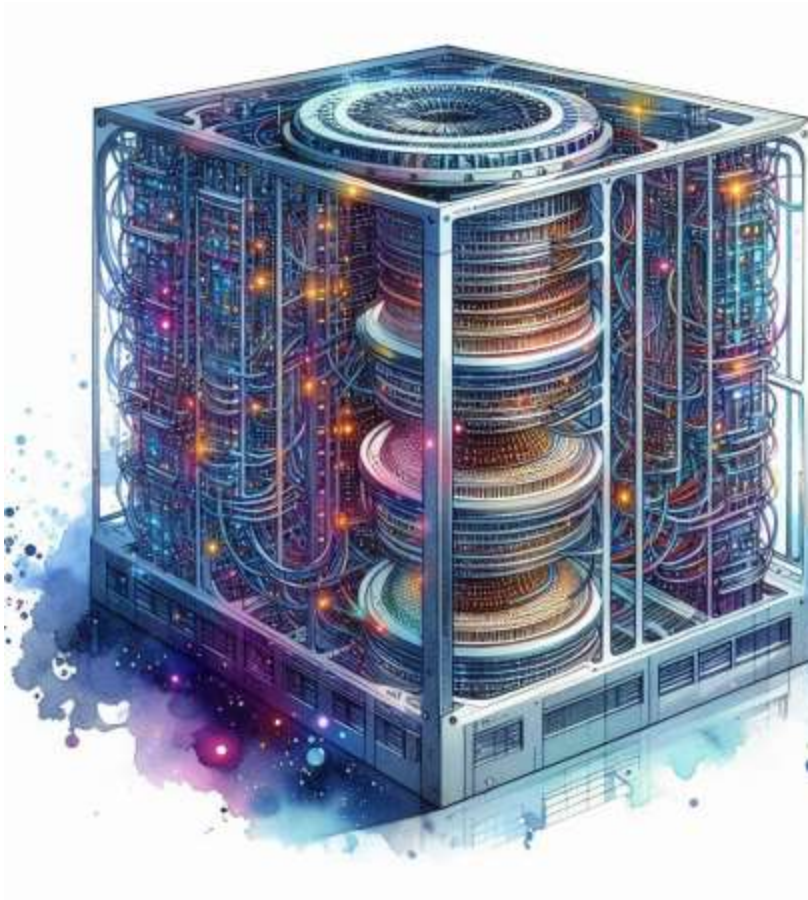
NetBeans es un entorno de desarrollo integrado, de código abierto y sigue la filosofía del software libre, y está hecho principalmente para el uso del lenguaje de programación Java.

- También existen muchísimos módulos extras para extender su funcionamiento.

  NetBeans es un producto libre y gratuito sin restricciones de uso y es un proyecto de código abierto de gran éxito con una gran base de usuarios y una comunidad en constante crecimiento.

SISTEMA OPERATIVO: WINDOWS 10

Windows 10 es un sistema operativo desarrollado por Microsoft y es la última versión de la familia de sistemas operativos Windows. Fue lanzado en julio de 2015 y ha sido una actualización importante en comparación con las versiones anteriores de Windows.



Proyecto1Analizador

1. Introducción

La clase `Proyecto1Analizador` es una aplicación en Java con una interfaz gráfica que permite al usuario analizar y optimizar código fuente, específicamente HTML, mediante un proceso de eliminación de espacios en blanco y comentarios, y generar archivos HTML que contienen reportes de análisis léxico. La interfaz se construye utilizando componentes de la librería Swing y se divide en áreas para el ingreso y visualización de código.

2. Requisitos Previos

Herramientas Necesarias:

- **Java Development Kit (JDK)**: Versión 17 o superior.
- **Entorno de Desarrollo Integrado (IDE)**: Eclipse, IntelliJ IDEA, NetBeans u otro.
- **Apache Maven** (Opcional): Si se usa para gestionar dependencias.

Dependencias:

- No existen dependencias externas definidas, salvo las bibliotecas estándar de Java, especialmente el paquete Swing para la interfaz gráfica.

3. Descripción General de la Clase

La clase principal `Proyecto1Analizador` consta de tres funcionalidades principales:

1. **Ingreso de código HTML**.
2. **Optimización del código eliminando espacios en blanco y comentarios**.
3. **Generación de un archivo HTML y su visualización**.

Componentes Clave:

- **`JTextArea` para ingresar código**: Permite al usuario introducir código fuente para analizar o optimizar.
- **`JTextArea` para código optimizado**: Muestra el código después de ser optimizado.
- **Botones**: Controlan las acciones de analizar, optimizar, y mostrar el archivo HTML generado.
- **Interfaz gráfica (GUI)**: La aplicación utiliza un `JSplitPane` para dividir la ventana en dos áreas (código de entrada y código optimizado) y organiza los botones en un panel inferior.

Organización de la Interfaz:

- **`JSplitPane`**: Divide la ventana en dos partes horizontales: el área de texto para ingresar el código y el área de texto para mostrar el código optimizado.
- **Panel de botones**: Situado en la parte inferior de la ventana, permite la interacción del usuario mediante tres botones principales: "Analizar Código", "Optimizar Código", y "Mostrar Página HTML".

4. Descripción del Código

4.1. Método `main`

Este método es el punto de entrada de la aplicación. Configura el **Look and Feel Nimbus** para darle una apariencia moderna a la interfaz gráfica y organiza los elementos de la GUI dentro de la ventana principal (`JFrame`).

- **Interfaz gráfica (`JFrame`)**: Se crea una ventana principal de tamaño 800x400 píxeles.
- **Componentes gráficos**: Se configuran dos áreas de texto: una para el ingreso del código fuente (`textArea`) y otra para mostrar el código optimizado (`optimizedTextArea`). Se utiliza un `JSplitPane` para dividir la ventana y permitir la redimensión dinámica.

4.2. Botones

1. **Botón "Analizar Código"**:
    - Al hacer clic, se toma el código fuente del área de texto (si ya fue optimizado, se usa la versión optimizada) y se procesa mediante la clase `ManejadorAnalizadores`. Esta clase realiza el análisis léxico del código y genera archivos HTML.
    - **Resultados**: Los tokens se imprimen y se generan dos archivos: uno con los tokens y otro con la página HTML.
2. **Botón "Optimizar Código"**:
    - Llama al método `eliminarEspaciosYComentarios` para eliminar comentarios y líneas en blanco del código fuente.
    - Actualiza el área de texto de código optimizado para reflejar los cambios.

3. **Botón "Mostrar Página HTML"**:
    - Muestra la página HTML generada en una nueva ventana (`JFrame`) utilizando un `JEditorPane` que carga y muestra el archivo HTML en un entorno no editable.

4.3. Método `eliminarEspaciosYComentarios`

Este método recibe una cadena de texto que representa el código fuente y realiza las siguientes operaciones:

- **Elimina comentarios de una sola línea** (aquellos que comienzan con `//`).
- **Elimina líneas vacías**.
- Devuelve una versión optimizada del código que se muestra en el área correspondiente.

4.4. Método `mostrarHTML`

Este método abre una nueva ventana para visualizar el archivo HTML generado. Utiliza un `JEditorPane` para cargar y mostrar el contenido de un archivo HTML en formato de página web.

5. Ejecución del Programa

Pasos para Ejecutar:

1. **Compilación y Ejecución**:
    - Compila el código en un IDE compatible con Java 17 o superior.
    - Ejecuta la aplicación, lo que abrirá la interfaz gráfica.
2. **Interacción con la Aplicación**:
    - Ingresa el código HTML en el área de texto.
    - Haz clic en el botón "Optimizar Código" para eliminar comentarios y espacios en blanco.
    - Haz clic en "Analizar Código" para generar los tokens y reportes en archivos HTML.
    - Haz clic en "Mostrar Página HTML" para visualizar el archivo HTML generado.

1. Descripción General

La clase `AnalizadorHTML` es un componente diseñado para procesar y analizar código fuente HTML, identificando y clasificando diversos elementos como etiquetas, atributos, comentarios, cadenas, signos, y otros componentes del lenguaje HTML. El analizador reconoce etiquetas tanto en inglés como en un conjunto predefinido de etiquetas traducidas al español, generando una lista de tokens para representar los elementos del código fuente.

2. Propiedades

- **`codigoFuente` (String):** Contiene el código fuente HTML que se va a analizar.
- **`pos` (int):** Indicador de la posición actual en el código fuente.
- **`linea` (int):** Rastrea la línea actual durante el análisis, útil para la localización de tokens.
- **`columna` (int):** Rastrea la columna actual dentro de la línea, utilizada para la ubicación precisa de cada token.
- **`tokens` (List<Token>):** Lista que almacena los tokens generados a partir del análisis del código fuente.
- **`palabrasReservadas` (Set<String>):** Conjunto de palabras reservadas de HTML que el analizador debe reconocer dentro de las etiquetas.
- **`TRADUCCION_ETIQUETAS` (Map<String, String>):** Diccionario estático que contiene la traducción de etiquetas HTML de español a inglés. Este mapa está preconfigurado para traducir etiquetas como "principal" a "main" y "navegacion" a "nav", entre otras.

3. Métodos Públicos

3.1. `AnalizadorHTML(String codigoFuente)`

Este es el constructor de la clase que inicializa los atributos con valores iniciales y configura las palabras reservadas que serán reconocidas en el análisis.

- **Parámetro:**
  - `codigoFuente`: El código fuente HTML que será analizado.

3.2. `List<Token> analizar()`

Este es el método principal que realiza el análisis del código fuente. Itera sobre cada carácter del código fuente y lo clasifica según su tipo (etiqueta, cadena, comentario, signo, etc.), creando un conjunto de tokens.

- **Devuelve:**
  - Una lista de objetos `Token` representando cada componente del código HTML.

3.3. `int getPosicionActual()`

Devuelve la posición actual del puntero en el código fuente.

4. Métodos Privados

4.1. `private void manejarTexto()`

Este método gestiona el texto que se encuentra entre etiquetas en el código HTML. Identifica si el texto está correctamente colocado dentro de una etiqueta o fuera de ella (lo que genera un error).

- Si el texto está entre etiquetas válidas, se crea un token del tipo "Texto".
- Si el texto está fuera de una etiqueta, se llama al método `manejarErrorTextoFueraDeEtiqueta()` para gestionar el error.

4.2. `private void manejarErrorTextoFueraDeEtiqueta()`

Este método maneja el caso de texto que aparece fuera de las etiquetas HTML, lo que no es permitido según el estándar. El texto es clasificado como un error y se genera un token de error correspondiente.

4.3. `private void manejarEtiqueta()`

Este método se activa cuando se encuentra un carácter `<` en el código fuente. Dependiendo del siguiente carácter (`/` o el nombre de una etiqueta), el método gestiona etiquetas de apertura o cierre.

4.4. `private void manejarEtiquetaApertura()`

Este método maneja etiquetas de apertura (`<nombreEtiqueta>`). Si la etiqueta es reconocida y válida, se genera un token correspondiente. Además, este método también gestiona los atributos dentro de la etiqueta.

4.5. `private void manejarEtiquetaCierre()`

Este método gestiona las etiquetas de cierre (`</nombreEtiqueta>`). Valida que la etiqueta cerrada sea correcta y genera el token de cierre correspondiente.

4.6. `private void manejarPalabrasReservadasDentroEtiqueta()`

Este método gestiona los atributos que aparecen dentro de una etiqueta HTML. Reconoce palabras reservadas (como `class`, `id`, etc.) y genera tokens para el nombre del atributo y su valor.

4.7. `private void manejarComentario()`

Este método gestiona los comentarios en el código HTML, capturando todo el contenido de un comentario que empieza con `//` hasta el final de la línea.

4.8. `private void manejarCadena()`

Este método gestiona las cadenas de texto delimitadas por comillas dobles (`"`). Captura todo el contenido entre las comillas y genera un token de tipo "Cadena".

4.9. `private boolean esEtiqueta(String lexema)`

Este método comprueba si una cadena de texto corresponde a una etiqueta válida, es decir, si se encuentra en el diccionario `TRADUCCION_ETIQUETAS`.

4.10. `private Token manejarSigno()`

Este método maneja los caracteres especiales en HTML (como `<`, `>`, `/`, etc.) que no forman parte de una etiqueta o comentario. Genera un token correspondiente al signo.

4.11. `private void avanzar()`

Este método avanza la posición actual del análisis (`pos`) y actualiza la línea y columna según sea necesario, considerando saltos de línea (`\n`).

5. Creación de Tokens

```
private Token crearToken(String lexema, String tipoLexema, String
traduccion, String lenguaje, String categoria, int linea, int columna)
```

Este método crea un objeto `Token` con la información provista:

- **`lexema`:** El texto que representa el token.
- **`tipoLexema`:** Tipo del lexema (por ejemplo, etiqueta, comentario, etc.).
- **`traduccion`:** Traducción del lexema, si aplica.
- **`lenguaje`:** El lenguaje (en este caso, HTML).
- **`categoria`:** La categoría del token (como apertura, cierre, etc.).
- **`linea`:** La línea donde fue encontrado el token.
- **`columna`:** La columna en la que fue encontrado el token.

6. Excepciones y Manejo de Errores

- **Texto fuera de etiquetas:** El método `manejarErrorTextoFueraDeEtiqueta()` se encarga de capturar y marcar como error cualquier texto que no esté contenido dentro de etiquetas válidas.

- **Etiquetas no reconocidas:** Si el analizador encuentra una etiqueta que no puede traducir o validar, genera un token de error que indica que la etiqueta no tiene traducción.

7. Traducción de Etiquetas

La clase contiene un diccionario predefinido (`TRADUCCION_ETIQUETAS`) que traduce etiquetas en español a su equivalente en inglés. Por ejemplo:

- `"principal"` se traduce a `"main"`.
- `"anclaje"` se traduce a `"a"`.

Este mapa permite que el analizador procese código HTML en español y lo traduzca automáticamente a inglés cuando sea necesario.

8. Consideraciones de Diseño

- **Extensibilidad:** Se puede expandir fácilmente añadiendo más palabras reservadas o etiquetas al diccionario de traducción.
- **Manejo de Errores:** El sistema marca los errores en la generación de tokens y permite continuar el análisis incluso en caso de encontrar texto malformado o etiquetas no reconocidas.

AnalizadorCSS

Introducción

El `AnalizadorCSS` es una clase Java diseñada para analizar código fuente CSS y generar una lista de tokens que representan elementos significativos del código. Este manual técnico proporciona una visión detallada de la estructura, funcionalidades y diseño del analizador.

Estructura del AnalizadorCSS

El `AnalizadorCSS` está diseñado para operar sobre un string que contiene código CSS. Utiliza métodos para reconocer y clasificar diferentes elementos del código, generando tokens que luego pueden ser utilizados para análisis posterior o para implementar funcionalidades de un editor o IDE.

Atributos Principales

- `codigoFuente`: Almacena el código CSS que se analizará.
- `pos`, `linea`, `columna`: Posiciones actuales en el código fuente para seguimiento y reporte de errores.

- `tokens`: Lista que almacena los tokens generados durante el análisis.

Métodos Principales

- `analizar()`: Método principal que inicia el análisis del código CSS. Itera sobre el código fuente caracter por caracter, invocando métodos específicos para manejar diferentes tipos de tokens.
- Métodos de Manejo de Tokens:
  - `manejarEspacio()`: Maneja secuencias de espacios en blanco.
  - `manejarNumero()`: Reconoce números enteros y decimales.
  - `manejarColorHexadecimal()`, `manejarColorRgba()`: Reconocen colores en formato hexadecimal y rgba respectivamente.
  - `etiquetaOReglaOIdentificador()`: Reconoce etiquetas HTML, reglas CSS y identificadores.
  - `manejarSelectorClase()`, `manejarID()`: Reconocen selectores de clase y IDs.
  - `manejarComentario()`: Reconoce y maneja comentarios CSS.
  - `manejarCadena()`: Reconoce y maneja cadenas delimitadas por comillas simples o dobles.
  - `manejarCombinador()`: Reconoce y maneja combinadores CSS como `>`, `+`, `~`, `.`
  - `manejarSigno()`, `manejarOtrosToken()`: Reconocen y manejan otros caracteres y signos especiales.

Métodos Auxiliares

- `avanzar()`: Avanza la posición actual en el código fuente, actualizando la línea y columna según sea necesario.
- `crearToken()`: Crea un objeto `Token` con la información del lexema, tipo, fila y columna.

Constantes y Arreglos

- `REGLAS_CSS`, `ETIQUETAS_HTML`, `OTROS`: Arreglos que contienen palabras clave y elementos reconocidos por el analizador.

Consideraciones de Diseño

El `AnalizadorCSS` está diseñado para ser eficiente y modular. Utiliza técnicas simples pero efectivas para reconocer tokens, como iteración sobre caracteres y comparaciones con expresiones regulares simples.

Uso del AnalizadorCSS

Para utilizar el `AnalizadorCSS`, se instancia con un string que contiene el código CSS a analizar. Luego, se llama al método `analizar()` para obtener la lista de tokens generados.

Estos tokens pueden ser utilizados para análisis sintáctico, resaltado de sintaxis, o cualquier otra funcionalidad que requiera el procesamiento estructurado del código CSS.

`AnalizadorJS`

Descripción General

La clase `AnalizadorJS` es un analizador léxico diseñado para procesar código fuente escrito en JavaScript. Esta clase identifica y clasifica diferentes tipos de tokens en el código, incluyendo identificadores, números, operadores, cadenas, comentarios y símbolos. El analizador es capaz de ignorar espacios en blanco y comentarios, facilitando así la extracción de información útil del código fuente.

Estructura de la Clase

Atributos

- **String codigoFuente**: Almacena el código fuente que será analizado.
- **int pos**: Representa la posición actual en el código fuente que está siendo analizada.
- **int linea**: Número de línea actual en el código fuente.
- **int columna**: Número de columna actual en la línea.
- **List<Token> tokens**: Lista que almacena todos los tokens identificados durante el análisis.

- **Parámetros**:
  - `codigoFuente`: Cadena que contiene el código JavaScript a analizar.
- **Funcionalidad**: Inicializa el analizador configurando el código fuente y estableciendo las posiciones iniciales de `pos`, `linea`, y `columna`, así como inicializando la lista de tokens.

Métodos Públicos

- **List<Token> analizar()**:
  - **Descripción**: Realiza el análisis léxico del código fuente. Itera a través del código, identificando y procesando diferentes tipos de tokens.
  - **Retorno**: Lista de tokens generados.
- **int getPosicionActual()**:
  - **Descripción**: Devuelve la posición actual en el código fuente.
  - **Retorno**: Posición actual (entero).

Métodos Privados

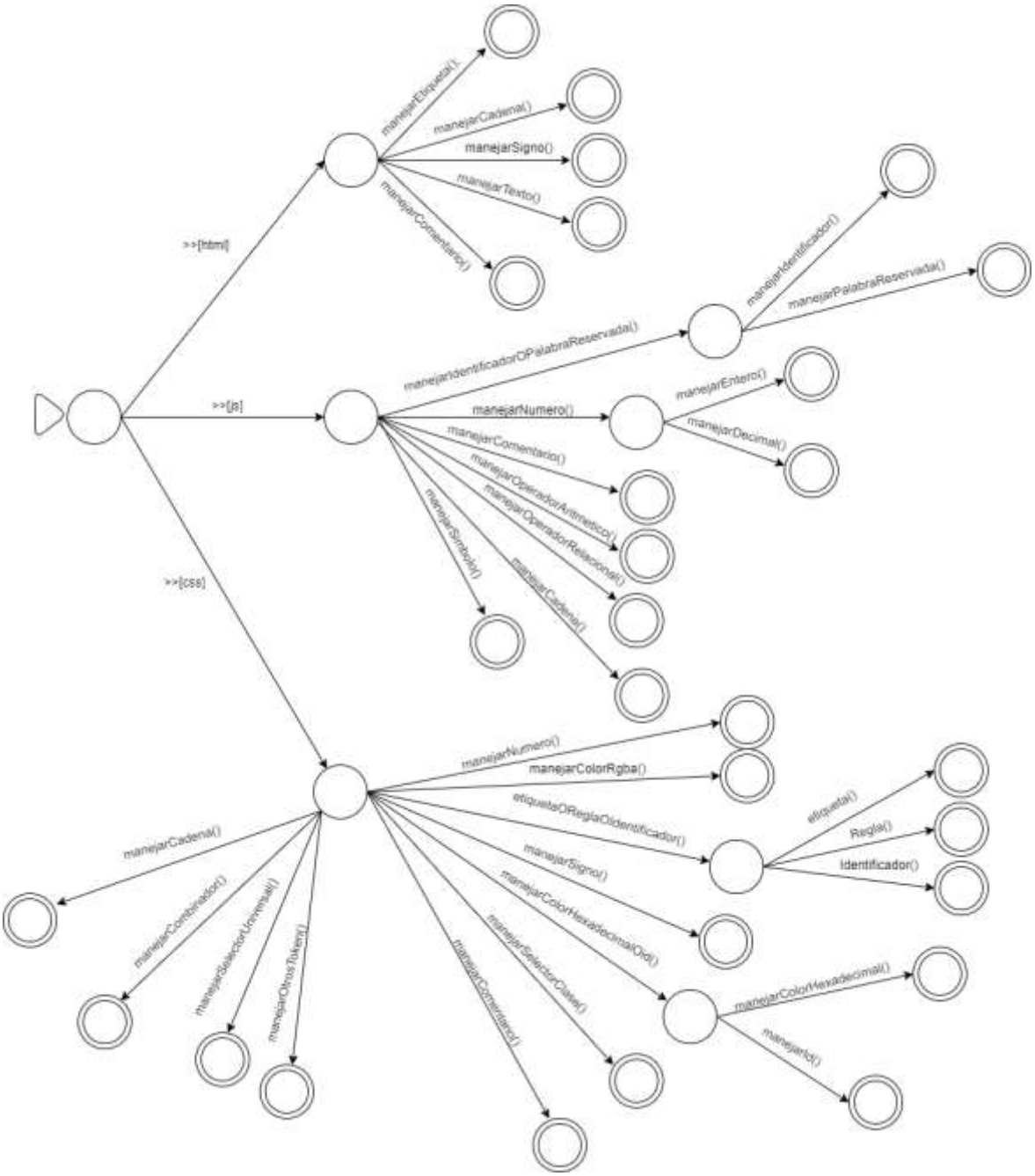
Los métodos privados manejan la lógica específica de procesamiento de tokens. A continuación se describen algunos de los más relevantes:

- **void manejarEspacio()**: Salta sobre los espacios en blanco en el código fuente.
- **void manejarIdentificadorOPalabraReservada()**: Identifica y maneja identificadores y palabras reservadas.
- **void manejarNumero()**: Procesa números enteros y decimales.
- **void manejarComentario()**: Maneja comentarios de una sola línea.
- **void manejarOperadorAritmetico(char actual)**: Procesa operadores aritméticos.
- **void manejarOperadorRelacional(char actual)**: Procesa operadores relacionales.
- **Token manejarCadena(char delimitador)**: Maneja cadenas de texto, incluyendo caracteres escapados.
- **void manejarSimbolo(char actual)**: Procesa símbolos como paréntesis, llaves, etc.

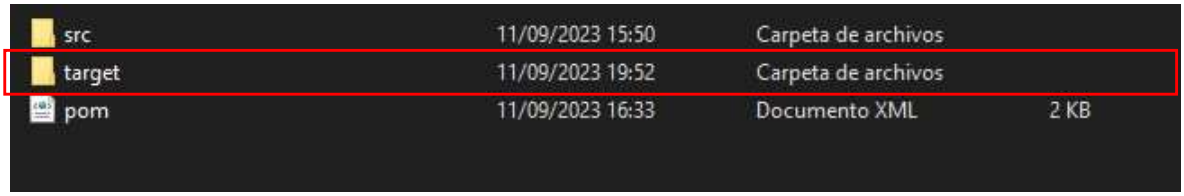
Métodos de Utilidad

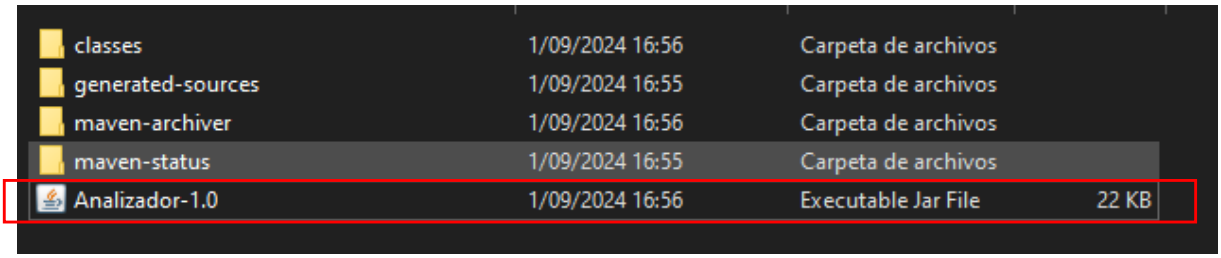
- **boolean esPalabraReservada(String lexema)**: Verifica si un lexema es una palabra reservada en JavaScript.
- **String obtenerNombreSimbolo(char actual)**: Devuelve una descripción textual del símbolo actual.
- **void avanzar()**: Avanza la posición actual en el código fuente, actualizando las posiciones de línea y columna según sea necesario.
- **Token crearToken(...)**: Crea un nuevo token a partir de los parámetros dados.

ARCHIVOS RESTANTES



ARCHIVOS JAR



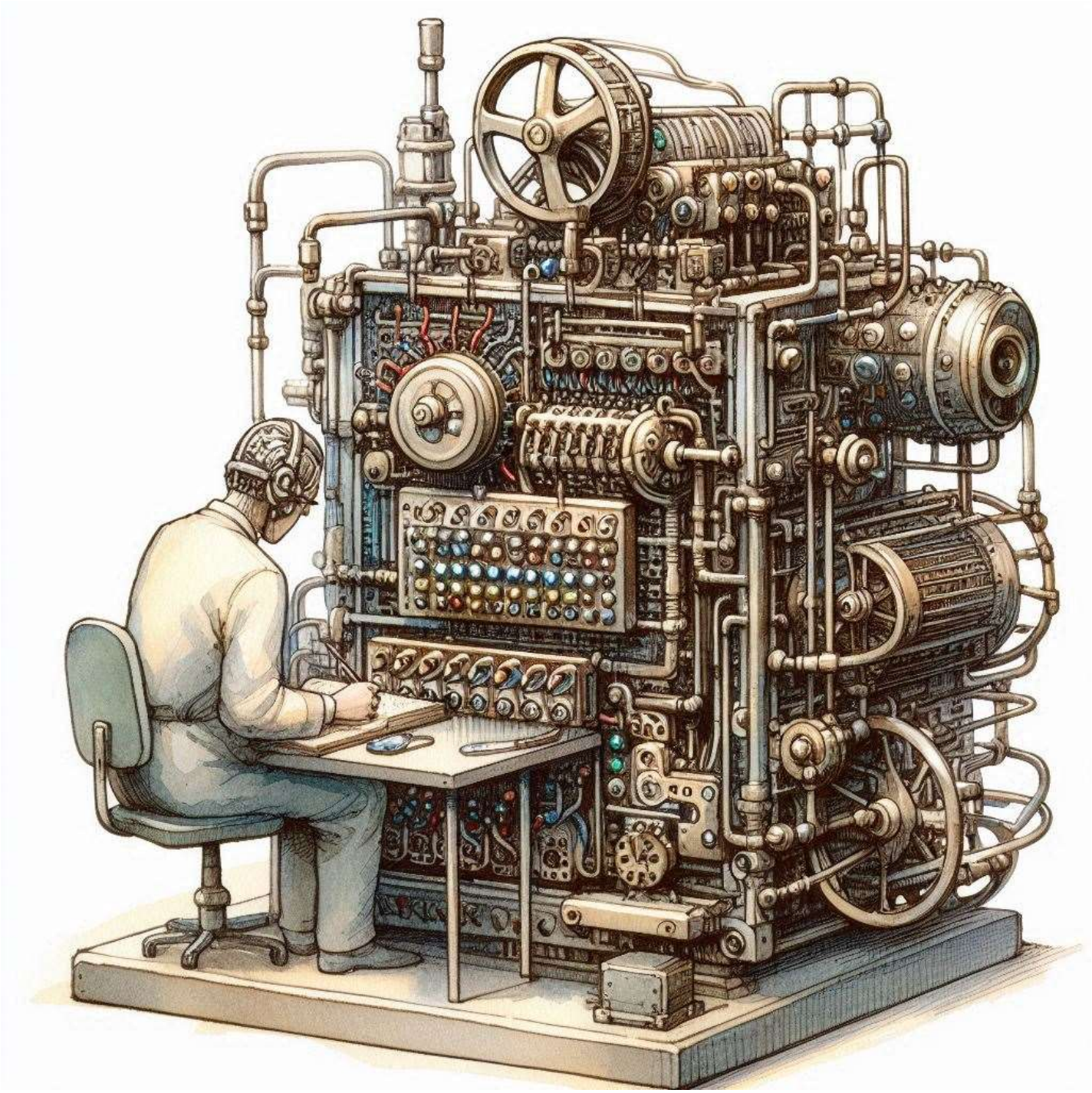


Repositorio

Link del repositorio:

https://github.com/Vega1402/Proyecto1Analizador

AUTOR



NOMBRE: VICTOR VEGA
CARNÉ: 202230772