# ANALIZADOR LEXICO

MANUAL TECNICO

## Presentación

El siguiente manual se ha desarrollado con la finalidad de dar a conocer la información necesaria para realizar mantenimiento, instalación y exploración del software para un Analizador Léxico, el cual consta de diferentes funcionalidades. El manual ofrece la información necesaria de ¿cómo está realizado el software? para que la persona (Desarrollador en JAVA) que quiera editar el software lo haga de una manera apropiada, dando a conocer la estructura del desarrollo del aplicativo.

## Resumen

El manual detalla los aspectos técnicos e informáticos del software para el Analizador Léxico con la finalidad de explicar la estructura del aplicativo al personal que quiera administrarlo, editarlo o configurarlo. La siguiente guía se encuentra dividida en las herramientas que se usaron para la creación del software con una breve explicación paso a paso. El aplicativo maneja diferentes funcionalidades los cuales se explicará que funcionamiento realiza cada uno de ellos, dando sugerencias para el debido uso del sistema de información.

## Introducción

El manual se realiza con el fin de detallar el software para el Analizador Léxico en términos técnicos para que la persona que vaya a administrar, editar o configurar el aplicativo lo haga de una manera apropiada. El documento se encuentra dividido en las siguientes secciones:

- ASPECTOS TEORICOS: Se darán a conocer conceptos, definiciones y explicaciones de los componentes del aplicativo desde un punto de vista teórico para mayor entendimiento por parte del lector sobre el funcionamiento del sistema de información y herramientas.
- DIAGRAMAS DE MODELAMIENTO: Se compone por diagramas e ilustraciones alusivos al funcionamiento del aplicativo.

## ASPECTOS TECNICOS

El aplicativo del Sistema Bancario tiene la finalidad de administrar Tarjetas que pertenezcan a determinados Clientes. Se recomienda que el siguiente manual sea manipulado únicamente por la persona que quiera administrar, editar o configurar el software para el Sistema Bancario para velar por la seguridad de los datos que se almacenan.

1. Herramientas utilizadas para el desarrollo: En ésta sección se procede a explicar las herramientas informáticas empleadas para el desarrollo del aplicativo:
   1.1. Apache NetBeans: NetBeans es un entorno de desarrollo integrado (IDE) para Java. NetBeans permite desarrollar aplicaciones a partir de un conjunto de componentes de software modulares llamados módulos. NetBeans se ejecuta en Windows, macOS, Linux y Solaris. Además del desarrollo en Java, cuenta con extensiones para otros lenguajes como PHP, C, C++, HTML5 y JavaScript. Las aplicaciones basadas en NetBeans, incluido NetBeans IDE, pueden ser ampliadas por desarrolladores externos.
   NetBeans IDE es un entorno de desarrollo integrado de código abierto. NetBeans IDE admite el desarrollo de todos los tipos de aplicaciones Java (Java SE (incluido JavaFX), Java ME, web, EJB y aplicaciones móviles) listas para usar. Entre otras características se encuentran un sistema de proyectos basado en Ant, soporte Maven, refactorizaciones, control de versiones (compatible con CVS, Subversion, Git, Mercurial y Clearcase). La plataforma Apache NetBeans es un marco genérico para aplicaciones Swing. Proporciona la "plomería" que, antes, cada desarrollador tenía que escribir por sí mismo: guardar el estado, conectar acciones a elementos del menú, elementos de la barra de herramientas y atajos de teclado; gestión de ventanas, etc.

## DIAGRAMAS DE MODELAMIENTO

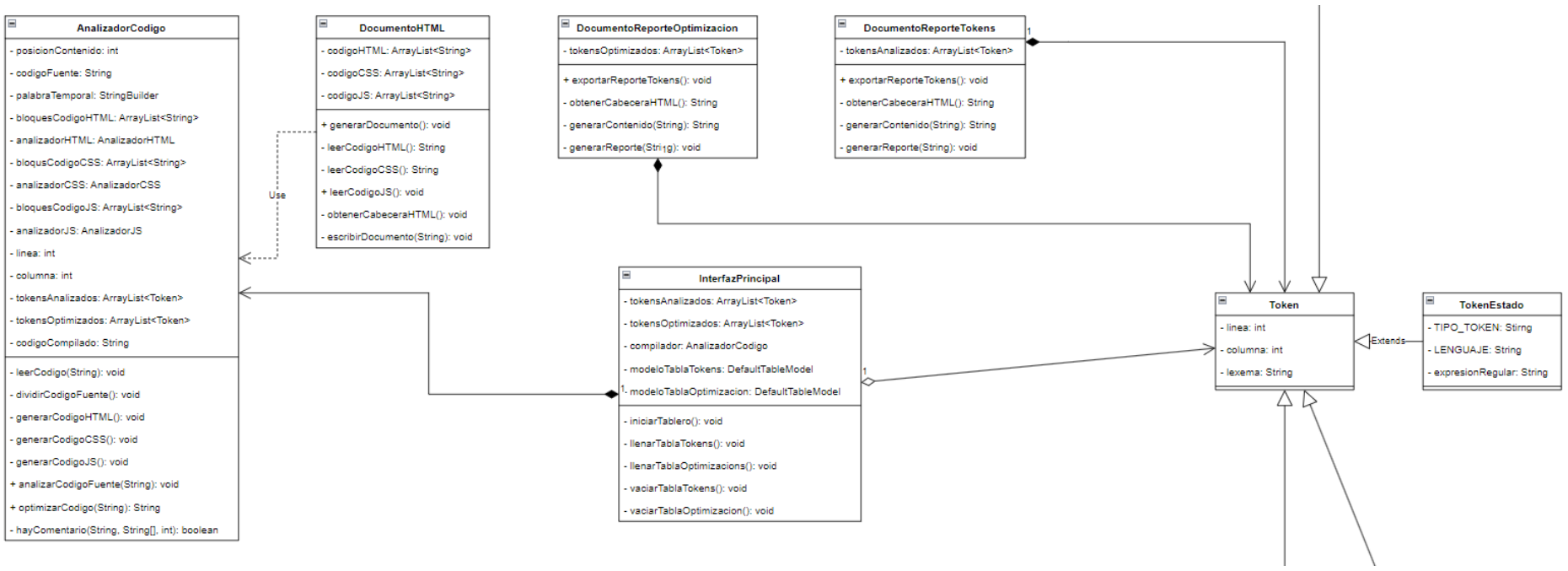
1. DIAGRAMA DE CLASES

El diagrama de clases está compuesto de las entidades, métodos y atributos que se crearon para el funcionamiento del software.

1.1. BACKEND

1.1.1. ANALIZADOR LEXICO GENERAL

En las siguientes imágenes se detalla cada una de las entidades (clases) usadas para el funcionamiento lógico del aplicativo, en el cual cada una de ellas realiza las funciones posteriormente descritas:



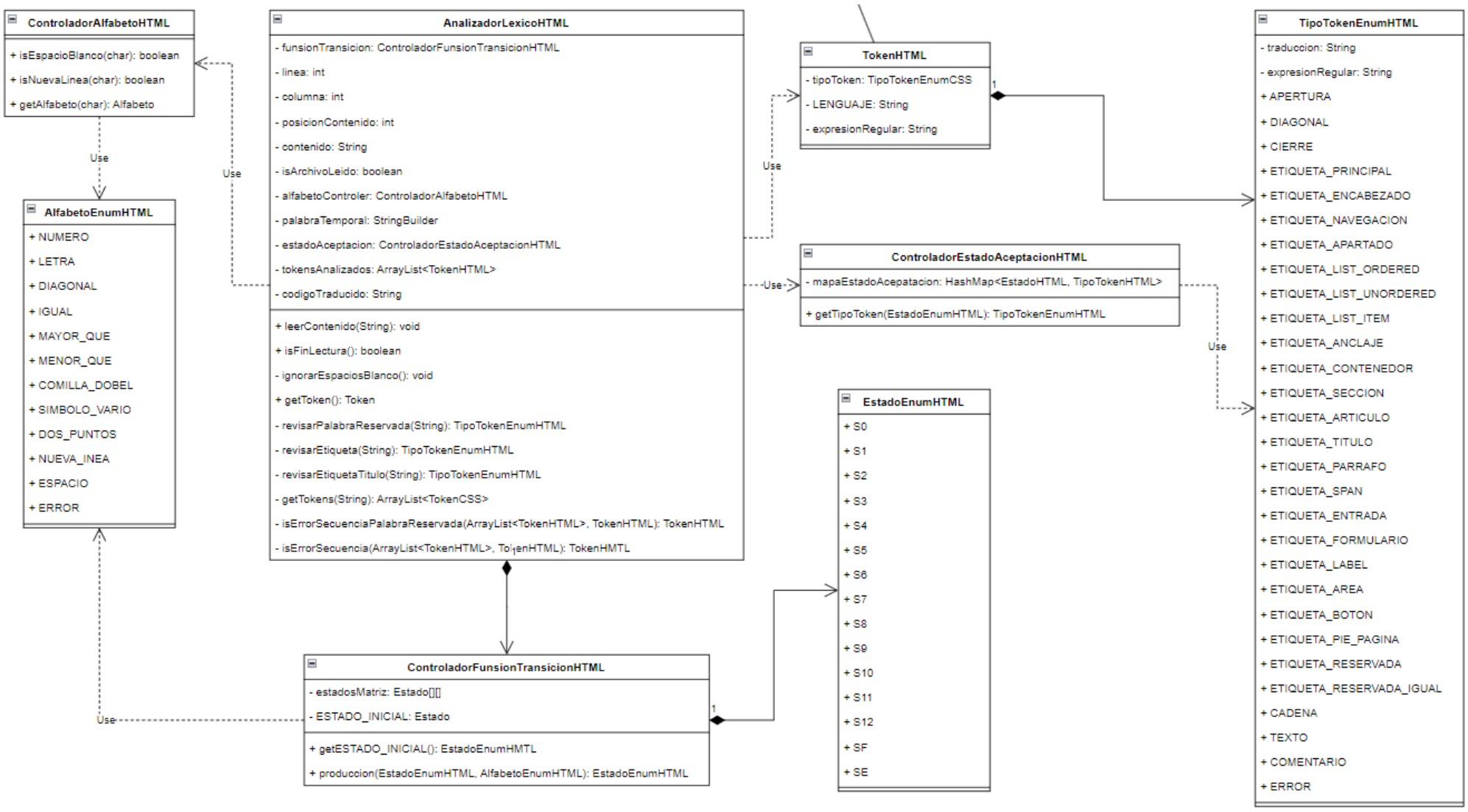

- **AnalizadorCodigo:** Clase que se encarga de realizar la lectura y análisis del código fuente que se haya ingresado en la aplicación, además también hace la verificación para que los tokens generados sean válidos para posterior uso con ellos.
  - leerCodigo: Este método lee el contenido recibido de manera que lo descompone byte a byte e inicializa algunos atributos que servirán para poder controlar la lectura del contenido ya recibido y descompuesto.
  - dividirCodigoFuente: Método que divide el código fuente total en bloques de código, según el token de cambio de estado para el lenguaje que corresponda analizar.

- agregarCodigoHTML: Método que analizada los bloques de código html del código fuente
- agrerarCodigoCSS: Método que analizada los bloques de código css del código fuente
- agrerarCodigoJS: Método que analizada los bloques de código java sript del código fuente
- analizarCodigoFuente: Método que invoca al método para leer el código fuente y posteriormente invoca al método para que se valla analizando el bloque de código correspondiente al lenguaje.
- optimizarCodigo: Método que se encarga de eliminar las líneas en blanco dentro del código fuente, y las líneas que contiene comentarios las "elimina" aguardando la línea para que posteriormente se analicen los tokens que se optimizaron en dicha línea de código.
- hayComentario: Método que verifica si en la línea "eliminada" con comentario, tenga en realidad un comentario valido que se deba de eliminar del código fuente ya optimizado.

- **DocumentoHTML:** Clase que se encarga de generar el archivo html funcional que representa al código compilado que se obtuvo del análisis del código fuente ingresado por el usuario en la aplicación
  - generarDocumento: Método que invoca a los métodos para poder obtener la cabecera del archivo html, y el contenido según los diferentes bloques de código hmtl, css y java script que se tengan en el código compilado, para que finalmente se pueda escribir y generar el archivo html funcional.
  - leerCodigoHTML: Método que lee el contenido de lenguaje html que se encuentra en el código ya compilado, y lo agrega a la data que se escribirá en el archivo final.
  - leerCodigoCSS: Método que lee el contenido de lenguaje css que se encuentra en el código ya compilado, y lo agrega a la data que se escribirá en el archivo final.
  - leerCodigoJS: Método que lee el contenido de lenguaje java script que se encuentra en el código ya compilado, y lo agrega a la data que se escribirá en el archivo final.
  - obtenerCabeceraHTML: Método que devuelve las etiquetas de cabecera que tendrá el archivo html final.
  - escribirDocumento: Método que genera y escribe el archivo html final que representa al código compilado del código fuente.
- **DocumentoReporteOptimizacion:** Clase que se encarga de generar un archivo html con el reporte de los tokens que se obtuvieron del análisis realizado con el código fuente ingresado por el usuario de la aplicación
  - exportarReporteTokens: Método que invoca a los métodos para obtener la cabecera del archivo html y el contenido según los diferentes tokens que se analizaron del código fuente, para que finalmente se pueda escribir y generar el archivo html de reporte de tokens
  - obtenerCabeceraHTML: Método que devuelve las etiquetas de cabecera que tendrá el archivo html final.
  - generarContenido: Método que va generando el contenido de la tabla que se mostrará en el archivo html final, según el contenido de los tokens que se analizaron del código fuente
  - generarReporte: Método que genera y escribe el archivo html final que contiene el reporte de los tokens analizados del código fuente.
- **DocumentoReporteTokens:** Clase que se encarga de generar un archivo html con el reporte de los tokens optimizados que se obtuvieron del análisis realizado con las líneas eliminadas que contenían comentarios del código fuente ingresado por el usuario de la aplicación
  - exportarReporteTokens: Método que invoca a los métodos para obtener la cabecera del archivo html y el contenido según los diferentes tokens optimizaron que se obtuvieron de las líneas comentadas eliminadas, para que finalmente se pueda escribir y generar el archivo html de reporte de tokens optimizados
  - obtenerCabeceraHTML: Método que devuelve las etiquetas de cabecera que tendrá el archivo html final.
  - generarContenido: Método que va generando el contenido de la tabla que se mostrará en el archivo html final, según el contenido de los tokens optimizaron que se obtuvieron de las líneas comentadas eliminadas
  - generarReporte: Método que genera y escribe el archivo html final que contiene el reporte de los tokens optimizaron que se obtuvieron de las líneas comentadas eliminadas
- **Token:** Clase que representa a un token el cual será generado por el Analizador Léxico cuando vaya analizando el código fuente de la Aplicación.
- **TokenEstado:** Es un tipo de Token que representa al token de cambio de estado, el cual no es más que un token representativo más dentro del código fuente que se analiza.

### 1.1.2. ANALIZADOR LEXICO DE HTML

En las siguientes imágenes se detalla cada una de las entidades (clases) usadas para el funcionamiento lógico del analizador léxico de html, en el cual cada una de ellas realiza las funciones posteriormente descritas:

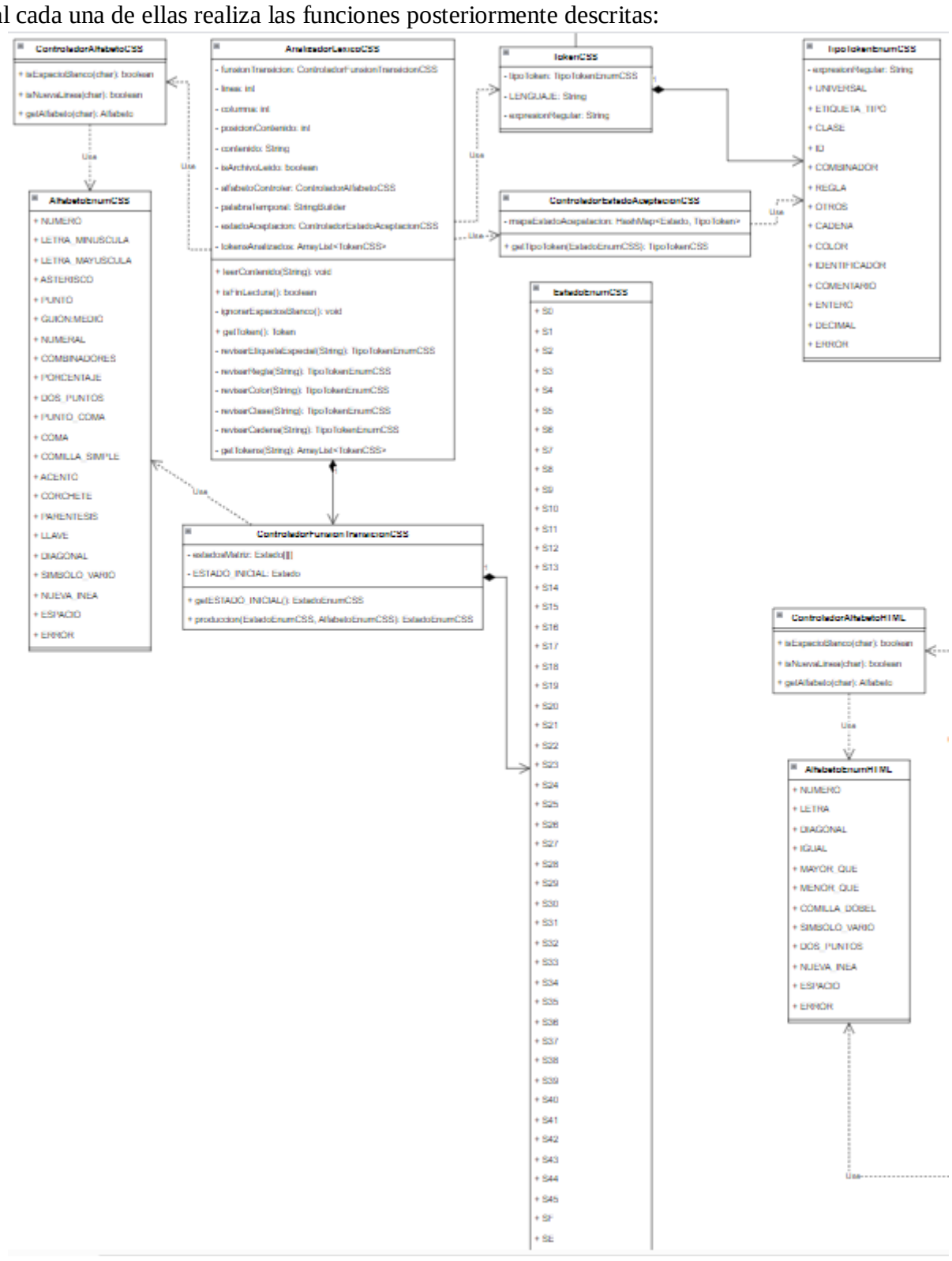



- **ControladorAlfabetoHTML:** Clase que contiene métodos para saber qué tipo de carácter se está manejando y de esa forma poder devolver si así lo fuera el alfabeto al que pertenece
    - isEspacioBlanco: Determina si el carácter recibido es un espacio en blanco dentro de los términos que tiene el lenguaje JAVA.
    - isNuevaLinea: Determina si el carácter recibido representa un salto de línea evaluando si el carácter es igual a \n
    - getAlfabeto: Devuelve un enumerado de tipo Alfabeto que dependerá del carácter recibido, el carácter se evalúa según su valor explícito o también según el valor que lo representa dentro del código ASCII.
- **AlfabetoEnumHTML:** Enumerado que contiene el alfabeto que se maneja dentro del Lenguaje para el Analizador Léxico.
- **AnalizadorLexicoHTML:** Clase que se encarga de realizar la lectura y análisis del código fuente que se haya ingresado en la aplicación, además también hace la verificación para que los tokens generados sean válidos para posterior uso con ellos.
    - leerContenido: Este método lee el contenido recibido de manera que lo descompone byte a byte e inicializa algunos atributos que servirán para poder controlar la lectura del contenido ya recibido y descompuesto.
    - isFinLectura: Determina si ya se llegó a la última posición del contenido leído.
    - ignorarEspacioBlanco: Método que lee un carácter para saber si el mismo es una nueva línea o es un espacio en blanco, de ser así sigue con el siguiente carácter, de lo contrario no sigue leyendo. Actualiza atributos que controlan la lectura, tales como la posición, fila y columna de lectura.
    - getToken: Método que se encarga de leer una palabra del contenido y de esa manera determinar y evaluar qué tipo de token se retornara junto con sus respectivos atributos de lectura, es decir, la fila y columna en las que fue leído y el contenido o palabra que lo representa.
    - revisarPalabraReservada: Método que verifica si la palabra recibida es igual a algunas de las que representa algún tipo de tokens ya establecidos para el lenguaje. De lo contrario devolverá *null*
    - revisarEtiqueta: Método que verifica si la palabra recibida es igual a algunas de las que representa algún tipo de tokens ya establecidos para el lenguaje. De lo contrario devolverá *null*
    - revisarEtiquetaTitulo: Método que verifica si la palabra recibida es igual a algunas de las que representa algún tipo de token para título ya establecidos para el lenguaje. De lo contrario devolverá *null*
    - getTokens: Método que obtienen todos los tokens que se lograron encontrar en el análisis del código fuente recibido.
    - isErrorSecuenciaPalabraReservada: Método que verifica que el token de tipo palabra reservada ingresado respete las reglas de las etiquetas de apertura y de cierre, si las respeta se devuelve el token tal y como esta, de lo contrario, el token devuelto será del tipo error.
    - isErrorSecuencia: Método que verifica que el token anterior al ingresado no sea de tipo error, si el token anterior es de tipo error entonces el token actual ingresado también lo será, de los contrario el token actual ingresado se mantiene igual.
- **ControladorFunsionTransicionHTML:** Clase que se encarga de llevar el control para hacer las transiciones de estados para los diferentes tipos de autómatas finitos deterministas manejados para el Analizador Léxico.
    - getEstadoInicial: Método que retorna el estado inicial para un nuevo autómata finito determinista
    - producción: Método que hace el cambio de estado en función del estado actual y el alfabeto que se reciben como parámetro.

- **ControladorEstadoAceptacionHTML:** Clase que se encarga de establecer el tipo de token que debe de corresponder según el último estado anterior al estado de aceptación en que haya terminado el autómata finito determinista que se estuvo manejando durante el análisis léxico del contenido.
    - getTipoToken: Método que retorna el tipo de token dependiendo del estado recibido como parámetro. Si el estado no tiene asignado un tipo de token en específico entonces se retornara un tipo de token error
- **EstadoEnumHTML:** Enumerado que contienen los estado de transición que se utilizaron para los diferentes Autómatas para el Analizador Léxico.
- **TokenHTML:** Clase que representa a un token el cual será generado por el Analizador Léxico cuando vaya analizando el código fuente de la Aplicación.
- **TipoTokenEnumHTML:** Enumerado que contiene todos los tipos de token que ir generando el Analizador Léxico.

### 1.1.3. ANALIZADOR LEXICO DE CSS

En las siguientes imágenes se detalla cada una de las entidades (clases) usadas para el funcionamiento lógico del analizador léxico de css, en el cual cada una de ellas realiza las funciones posteriormente descritas:

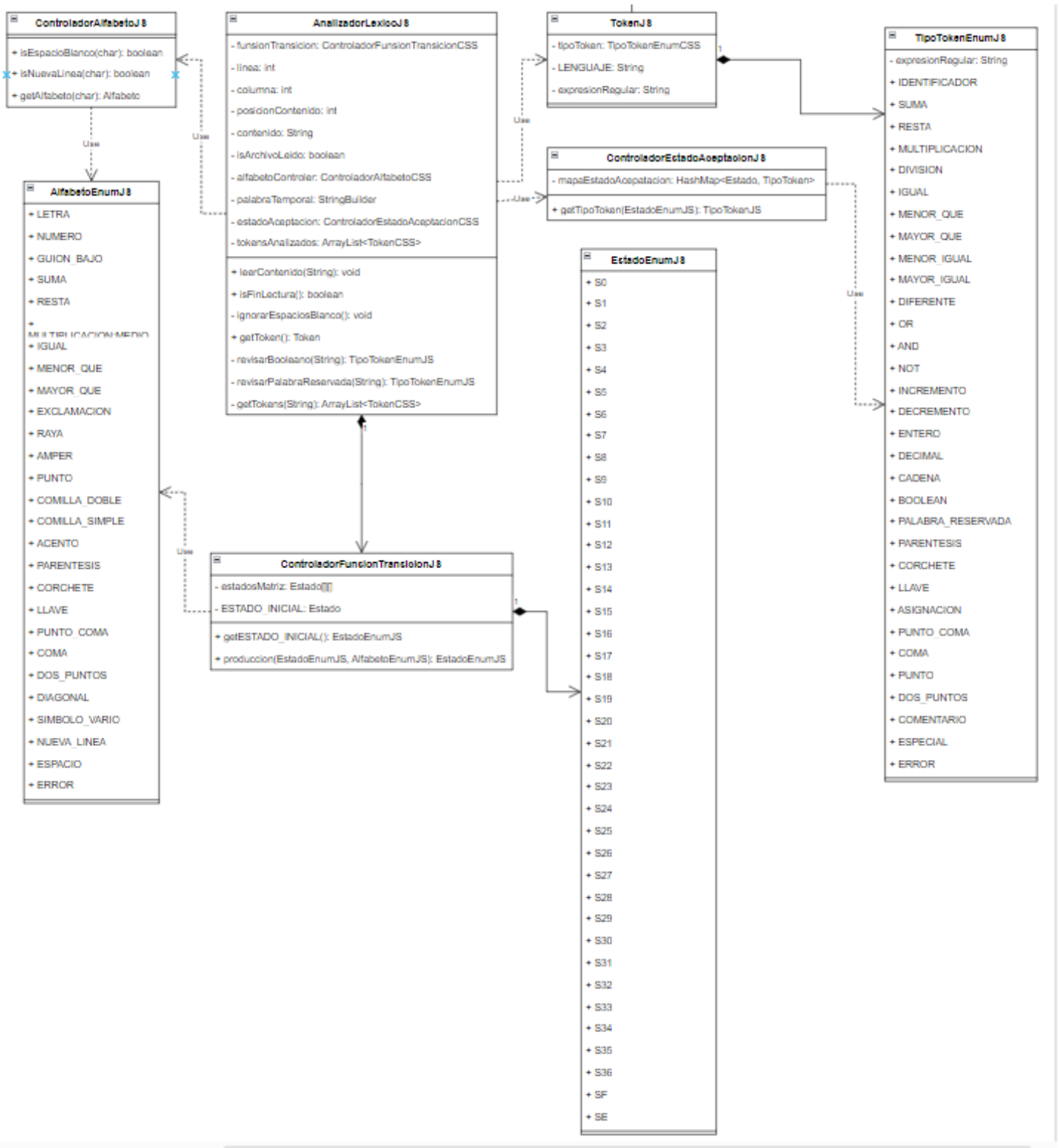



- **ControladorAlfabetoCSS:** Clase que contiene métodos para saber qué tipo de carácter se está manejando y de esa forma poder devolver si así lo fuera el alfabeto al que pertenece
    - isEspacioBlanco: Determina si el carácter recibido es un espacio en blanco dentro de los términos que tiene el lenguaje JAVA.
    - isNuevaLinea: Determina si el carácter recibido representa un salto de línea evaluando si el carácter es igual a \n
    - getAlfabeto: Devuelve un enumerado de tipo Alfabeto que dependerá del carácter recibido, el carácter se evalúa según su valor explícito o también según el valor que lo representa dentro del código ASCII.
- **AlfabetoEnumCSS:** Enumerado que contiene el alfabeto que se maneja dentro del Lenguaje para el Analizador Léxico.
- **AnalizadorLexicoCSS:** Clase que se encarga de realizar la lectura y análisis del código fuente que se haya ingresado en la aplicación, además también hace la verificación para que los tokens generados sean válidos para posterior uso con ellos.
    - leerContenido: Este método lee el contenido recibido de manera que lo descompone byte a byte e inicializa algunos atributos que servirán para poder controlar la lectura del contenido ya recibido y descompuesto.
    - isFinLectura: Determina si ya se llegó a la última posición del contenido leído.
    - ignorarEspacioBlanco: Método que lee un carácter para saber si el mismo es una nueva línea o es un espacio en blanco, de ser así sigue con el siguiente carácter, de lo contrario no sigue leyendo. Actualiza atributos que controlan la lectura, tales como la posición, fila y columna de lectura.

    - getToken: Método que se encarga de leer una palabra del contenido y de esa manera determinar y evaluar qué tipo de token se retornara junto con sus respectivos atributos de lectura, es decir, la fila y columna en las que fue leído y el contenido o palabra que lo representa.
    - revisarEtiquetaTipo: Método que verifica si la palabra recibida es igual a algunas de las que representa algún tipo de tokens ya establecidos para el lenguaje. De lo contrario devolverá *null*
    - revisarRegla: Método que verifica si la palabra recibida es igual a algunas de las que representa algún tipo de tokens ya establecidos para el lenguaje. De lo contrario devolverá *null*
    - revisarOtros: Método que verifica si la palabra recibida es igual a algunas de las que representa algún tipo de tokens ya establecidos para el lenguaje. De lo contrario devolverá *null*
    - revisarColor: Método que verifica si la palabra recibida es igual a algunas de las que representa algún tipo de tokens ya establecidos para el lenguaje. De lo contrario devolverá *null*
    - revisarClase: Método que verifica si la palabra recibida empieza con un punto que representa al token de tipo clase ya establecido para el lenguaje. De lo contrario devolverá *null*
    - revisarCadena: Método que verifica si la palabra recibida empieza con una comilla y corchete que representa al token de tipo cadena ya establecido para el lenguaje. De lo contrario devolverá *null*
    - getTokens: Método que obtienen todos los tokens que se lograron encontrar en el análisis del código fuente recibido.
- **ControladorFunsionTransicionCSS:** Clase que se encarga de llevar el control para hacer las transiciones de estados para los diferentes tipos de autómatas finitos deterministas manejados para el Analizador Léxico.
    - getEstadoInicial: Método que retorna el estado inicial para un nuevo autómata finito determinista
    - producción: Método que hace el cambio de estado en función del estado actual y el alfabeto que se reciben como parámetro.
- **ControladorEstadoAceptacionCSS:** Clase que se encarga de establecer el tipo de token que debe de corresponder según el último estado anterior al estado de aceptación en que haya terminado el autómata finito determinista que se estuvo manejando durante el análisis léxico del contenido.
    - getTipoToken: Método que retorna el tipo de token dependiendo del estado recibido como parámetro. Si el estado no tiene asignado un tipo de token en específico entonces se retornara un tipo de token error
- **EstadoEnumCSS:** Enumerado que contienen los estado de transición que se utilizaron para los diferentes Autómatas para el Analizador Léxico.
- **TokenCSS:** Clase que representa a un token el cual será generado por el Analizador Léxico cuando vaya analizando el código fuente de la Aplicación.
- **TipoTokenEnumCSS:** Enumerado que contiene todos los tipos de token que ir generando el Analizador Léxico.

### 1.1.4. ANALIZADOR LEXICO DE JS

En las siguientes imágenes se detalla cada una de las entidades (clases) usadas para el funcionamiento lógico del analizador léxico de java script, en el cual cada una de ellas realiza las funciones posteriormente descritas:

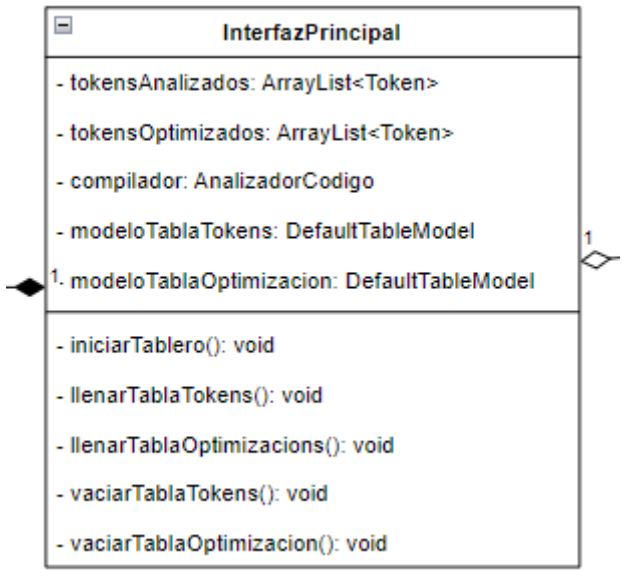



- **ControladorAlfabetoJS:** Clase que contiene métodos para saber qué tipo de carácter se está manejando y de esa forma poder devolver si así lo fuera el alfabeto al que pertenece
  - isEspacioBlanco: Determina si el carácter recibido es un espacio en blanco dentro de los términos que tiene el lenguaje JAVA.
  - isNuevaLinea: Determina si el carácter recibido representa un salto de línea evaluando si el carácter es igual a \n
  - getAlfabeto: Devuelve un enumerado de tipo Alfabeto que dependerá del carácter recibido, el carácter se evalúa según su valor explícito o también según el valor que lo representa dentro del código ASCII.
- **AlfabetoEnumJS:** Enumerado que contiene el alfabeto que se maneja dentro del Lenguaje para el Analizador Léxico.
- **AnalizadorLexicoJS:** Clase que se encarga de realizar la lectura y análisis del código fuente que se haya ingresado en la aplicación, además también hace la verificación para que los tokens generados sean válidos para posterior uso con ellos.
  - leerContenido: Este método lee el contenido recibido de manera que lo descompone byte a byte e inicializa algunos atributos que servirán para poder controlar la lectura del contenido ya recibido y descompuesto.
  - isFinLectura: Determina si ya se llegó a la última posición del contenido leído.
  - ignorarEspacioBlanco: Método que lee un carácter para saber si el mismo es una nueva línea o es un espacio en blanco, de ser así sigue con el siguiente carácter, de lo contrario no sigue leyendo. Actualiza atributos que controlan la lectura, tales como la posición, fila y columna de lectura.
  - getToken: Método que se encarga de leer una palabra del contenido y de esa manera determinar y evaluar qué tipo de token se retornara junto con sus respectivos atributos de lectura, es decir, la fila y columna en las que fue leído y el contenido o palabra que lo representa.
  - revisarBooleano: Método que verifica si la palabra recibida es igual a algunas de las que representa algún tipo de tokens ya establecidos para el lenguaje. De lo contrario devolverá *null*
  - revisarPalabraReservada: Método que verifica si la palabra recibida es igual a algunas de las que representa algún tipo de tokens ya establecidos para el lenguaje. De lo contrario devolverá *null*
  - getTokens: Método que obtienen todos los tokens que se lograron encontrar en el análisis del código fuente recibido.

- **ControladorFunsionTransicionJS:** Clase que se encarga de llevar el control para hacer las transiciones de estados para los diferentes tipos de autómatas finitos deterministas manejados para el Analizador Léxico.
    - getEstadoInicial: Método que retorna el estado inicial para un nuevo autómata finito determinista
    - producción: Método que hace el cambio de estado en función del estado actual y el alfabeto que se reciben como parámetro.
- **ControladorEstadoAceptacionJS:** Clase que se encarga de establecer el tipo de token que debe de corresponder según el último estado anterior al estado de aceptación en que haya terminado el autómata finito determinista que se estuvo manejando durante el análisis léxico del contenido.
    - getTipoToken: Método que retorna el tipo de token dependiendo del estado recibido como parámetro. Si el estado no tiene asignado un tipo de token en específico entonces se retornara un tipo de token error
- **EstadoEnumJS:** Enumerado que contienen los estado de transición que se utilizaron para los diferentes Autómatas para el Analizador Léxico.
- **TokenJS:** Clase que representa a un token el cual será generado por el Analizador Léxico cuando vaya analizando el código fuente de la Aplicación.
- **TipoTokenEnumJS:** Enumerado que contiene todos los tipos de token que ir generando el Analizador Léxico.

1.2. FORNTEND

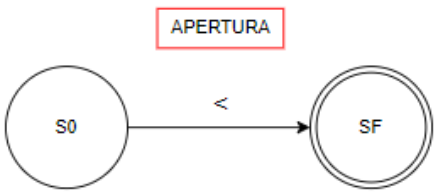
En las siguientes imágenes se detalla la entidad (clase) usadas para el funcionamiento visual del aplicativo, en el cual cada una de ellas realiza las funciones posteriormente descritas:

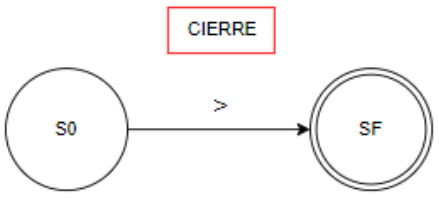


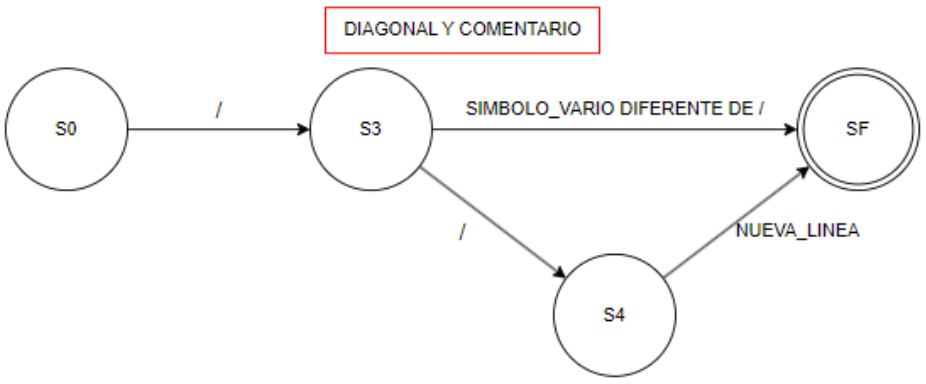

- **InterfazPrincipal:**
    - iniciarTablero: Método que le da un modelo especifico a las tablas para los reportes, para que se puedan mostrar los datos de forma correcta
    - llenarTablaTokens: Método que rellena la tabla para los reportes de tokens con los datos del arreglo que contiene todos las tokens que se generaron tras el análisis
    - llenarTablaOptimizacion: Método que rellena la tabla para los reportes de tokens optimizados con los datos del arreglo que contiene todos las tokens que se generaron tras el análisis
    - vaciarTablaTokens: Método que elimina los datos que estén existentes en la tabla de los reportes de tokens, para que no haya problemas con colapsos de datos
    - vaciarTablaOptimizacion: Método que elimina los datos que estén existentes en la tabla de los reportes de optimización, para que no haya problemas con colapsos de datos

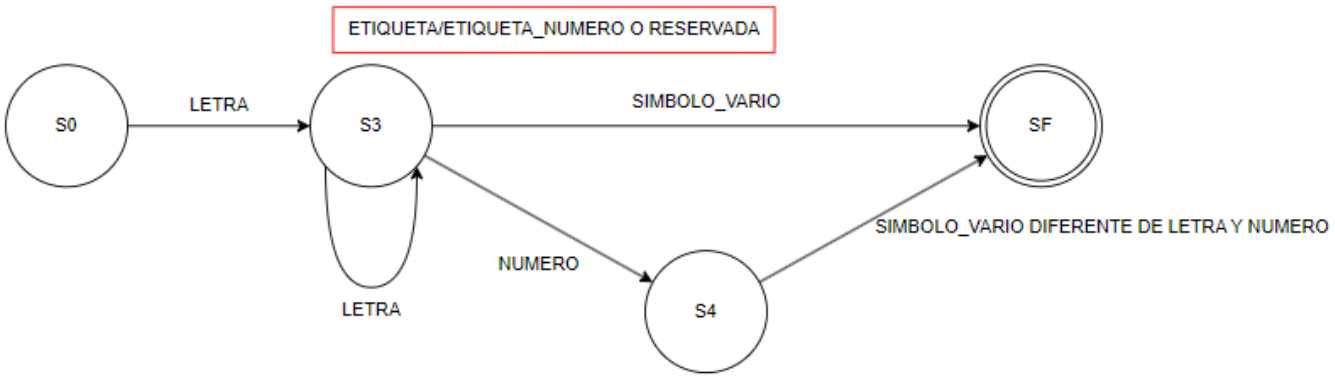
2. AUTOMATAS

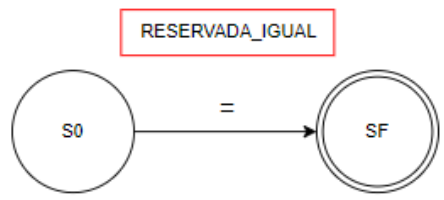
2.1. HTML

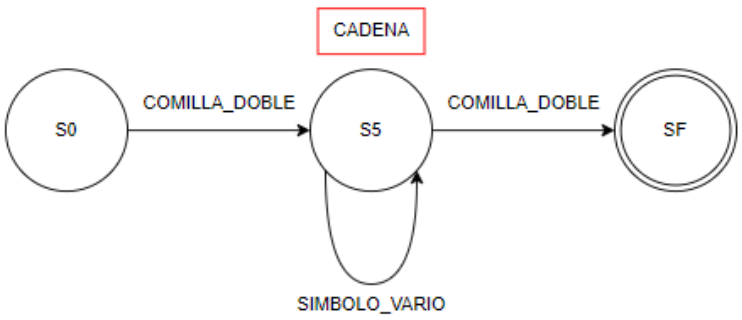


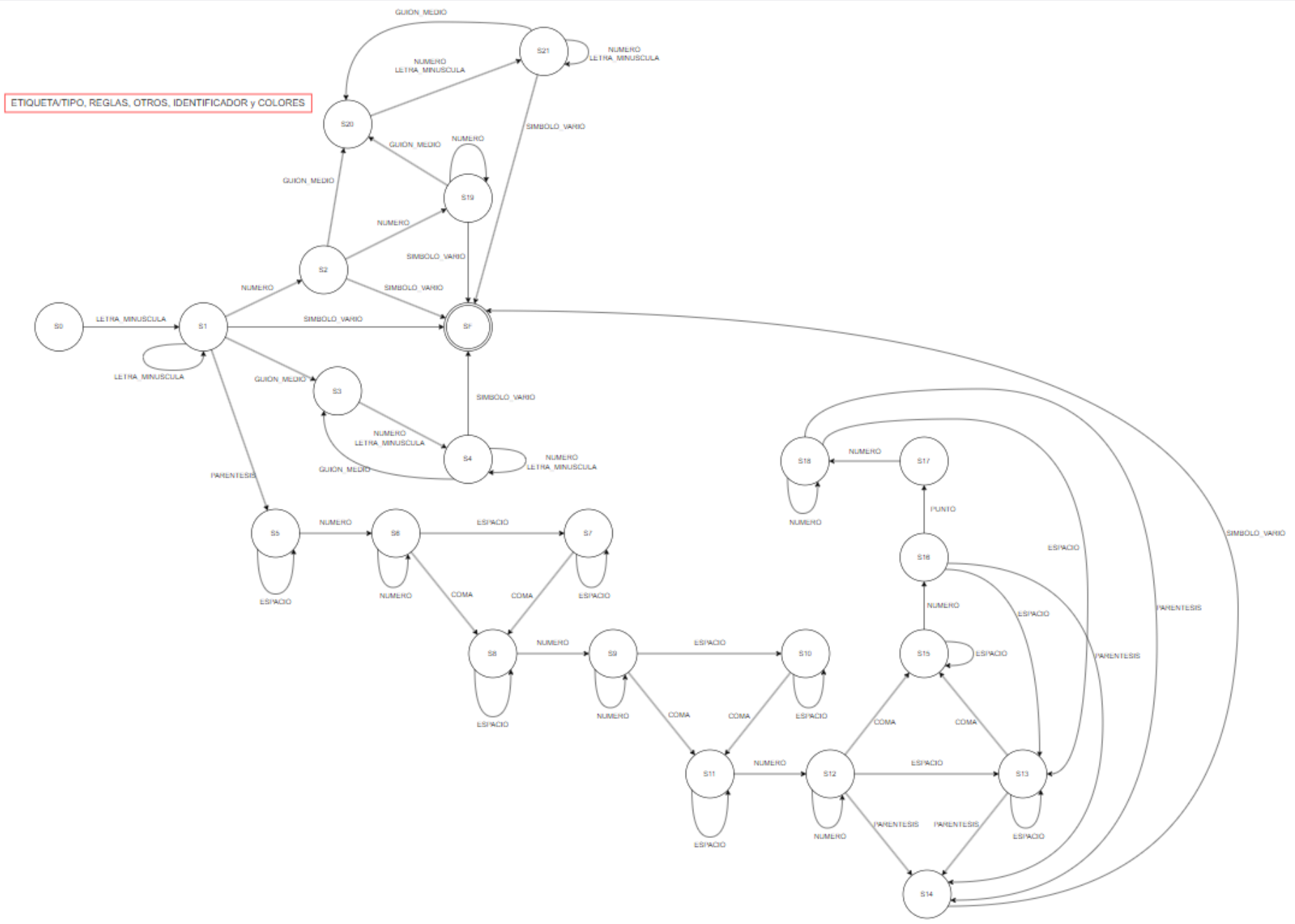



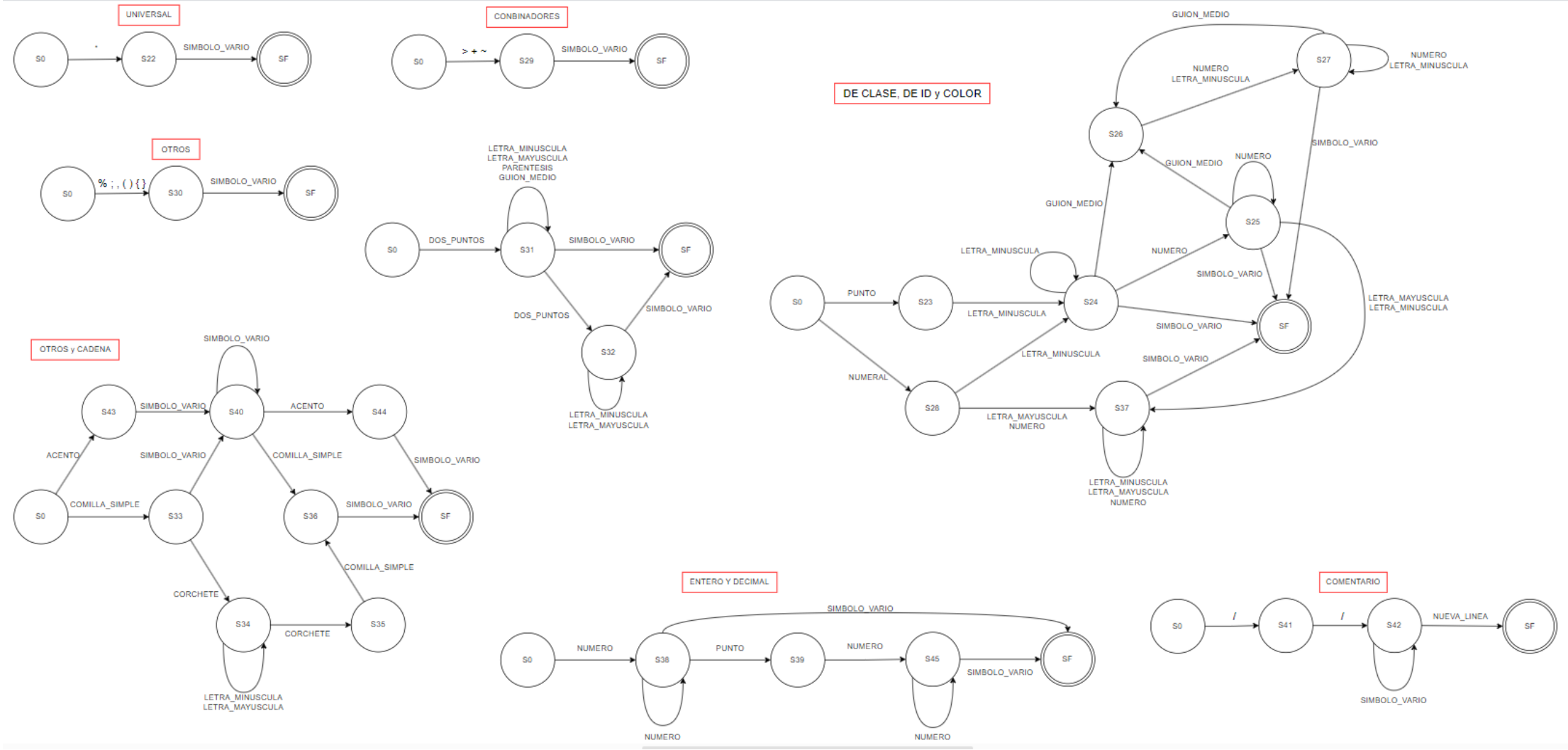




2.2. CSS






## 2.3. JAVA SCRIPT