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.

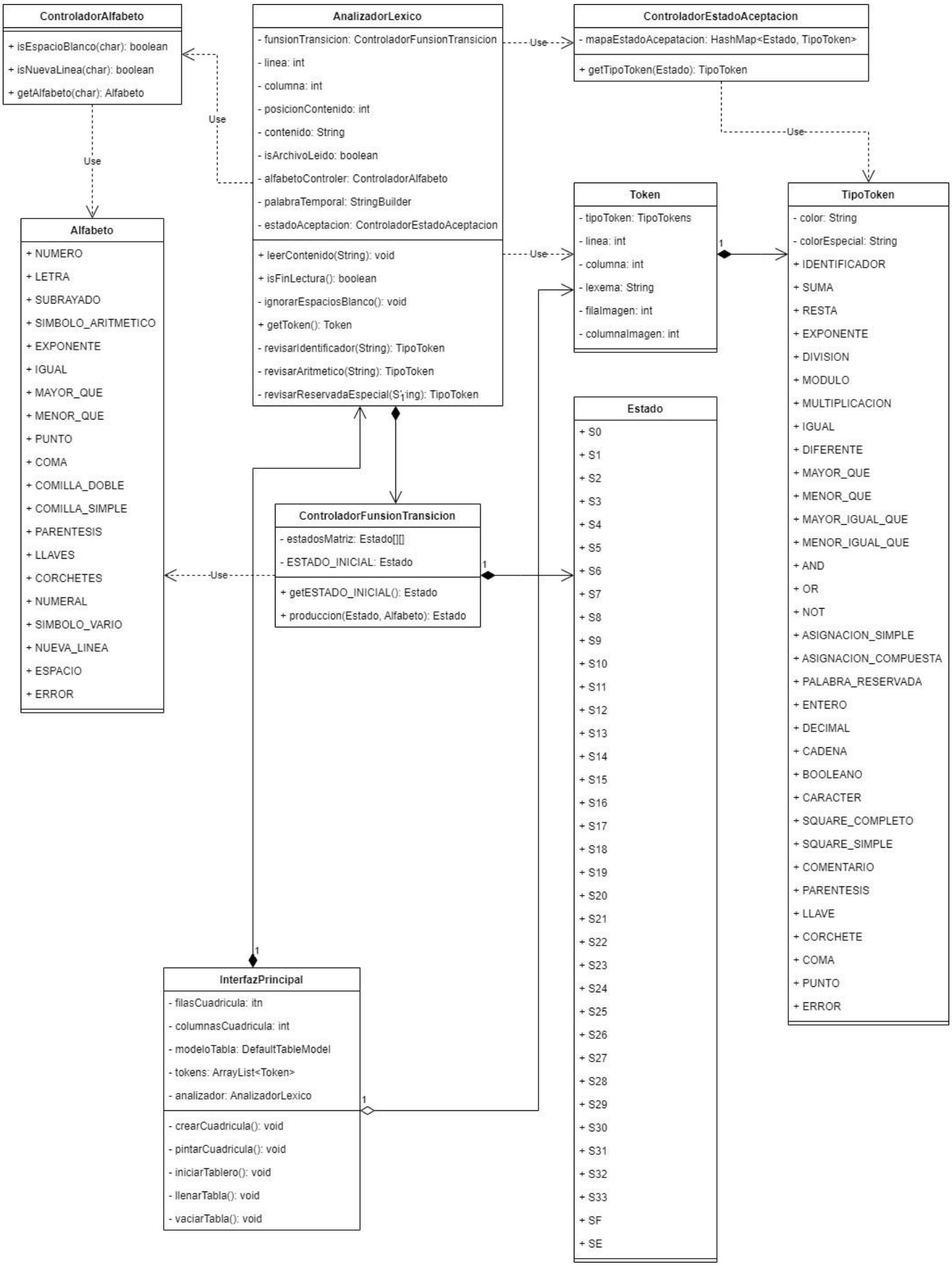
DIAGRAMAS DE MODELAMIENTO

1. DIAGRAMA DE CLASES

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

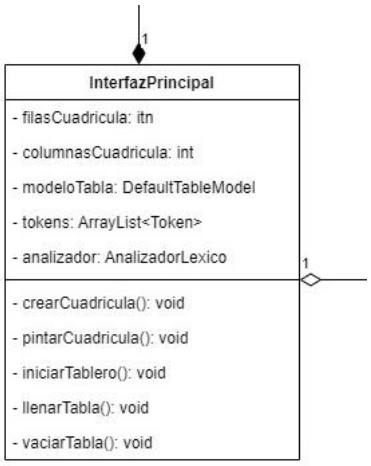
BackEnd

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:


ControladorAlfabeto
+ isEspacioBlanco(char): boolean
+ isNuevaLinea(char): boolean
+ getAlfabeto(char): Alfabeto
Use
Use
AnalizadorLexico
- funsionTransicion: ControladorFunsionTransicion
- linea: int
- columna: int
- posicionContenido: int
- contenido: String
- isArchivoLeido: boolean
- alfabetoControler: ControladorAlfabeto
- palabraTemporal: StringBuilder
- estadoAceptacion: ControladorEstadoAceptacion
+ leerContenido(String): void
+ isFinLectura(): boolean
- ignorarEspaciosBlanco(): void
+ getToken(): Token
- revisarIdentificador(String): TipoToken
- revisarAritmetico(String): TipoToken
- revisarReservadaEspecial(String): TipoToken
Use
ControladorEstadoAceptacion
- mapaEstadoAcepatacion: HashMap<Estado, TipoToken>
+ getTipoToken(Estado): TipoToken
Use
Alfabeto
+ NUMERO
+ LETRA
+ SUBRAYADO
+ SIMBOLO_ARITMETICO
+ EXPONENTE
+ IGUAL
+ MAYOR_QUE
+ MENOR_QUE
+ PUNTO
+ COMA
+ COMILLA_DOBLE
+ COMILLA_SIMPLE
+ PARENTESIS
+ LLAVES
+ CORCHETES
+ NUMERAL
+ SIMBOLO_VARIO
+ NUEVA_LINEA
+ ESPACIO
+ ERROR
Token
- tipoToken: TipoTokens
- linea: int
- columna: int
- lexema: String
- filaImagen: int
- columnaImagen: int
1
TipoToken
- color: String
- colorEspecial: String
+ IDENTIFICADOR
+ SUMA
+ RESTA
+ EXPONENTE
+ DIVISION
+ MODULO
+ MULTIPLICACION
+ IGUAL
+ DIFERENTE
+ MAYOR_QUE
+ MENOR_QUE
+ MAYOR_IGUAL_QUE
+ MENOR_IGUAL_QUE
+ AND
+ OR
+ NOT
+ ASIGNACION_SIMPLE
+ ASIGNACION_COMPUESTA
+ PALABRA_RESERVADA
+ ENTERO
+ DECIMAL
+ CADENA
+ BOOLEANO
+ CARACTER
+ SQUARE_COMPLETO
+ SQUARE_SIMPLE
+ COMENTARIO
+ PARENTESIS
+ LLAVE
+ CORCHETE
+ COMA
+ PUNTO
+ ERROR
1
Estado
+ S0
+ S1
+ S2
+ S3
+ S4
+ S5
+ S6
+ S7
+ S8
+ S9
+ S10
+ S11
+ S12
+ S13
+ S14
+ S15
+ S16
+ S17
+ S18
+ S19
+ S20
+ S21
+ S22
+ S23
+ S24
+ S25
+ S26
+ S27
+ S28
+ S29
+ S30
+ S31
+ S32
+ S33
+ SF
+ SE
ControladorFunsionTransicion
- estadosMatriz: Estado[][]
- ESTADO_INICIAL: Estado
+ getESTADO_INICIAL(): Estado
+ produccion(Estado, Alfabeto): Estado
Use
1
1
InterfazPrincipal
- filasCuadricula: itn
- columnasCuadricula: int
- modeloTabla: DefaultTableModel
- tokens: ArrayList<Token>
- analizador: AnalizadorLexico
- crearCuadricula(): void
- pintarCuadricula(): void
- iniciarTablero(): void
- llenarTabla(): void
- vaciarTabla(): void
1
1

- **ControladorAlfabeto:** 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.
- **Alfabeto:** Enumerado que contiene el alfabeto que se maneja dentro del Lenguaje para el Analizador Léxico.
- **AnalizadorLexico:** 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.
    - revisarIdentificador: 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*
    - revisarAritmetico: Método que verifica si la palabra recibida es igual a algún signo aritmético tales como +, -, * y /, si coincide con alguno de los anteriores signo entonces se devuelve un tipo de token que represente al signo. De lo contrario devolverá *null*
    - revisarReservadaEspecial: 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*
- **ControladorFunsionTransicion:** 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.

- **ControladorEstadoAceptacion:** 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
- **Estado:** Enumerado que contienen los estado de transición que se utilizaron para los diferentes Autómatas para el Analizador Léxico.
- **Token:** Clase que representa a un token el cual será generado por el Analizador Léxico cuando vaya analizando el código fuente de la Aplicación.
- **TipoToken:** Enumerado que contiene todos los tipos de token que ir generando el Analizador Léxico.

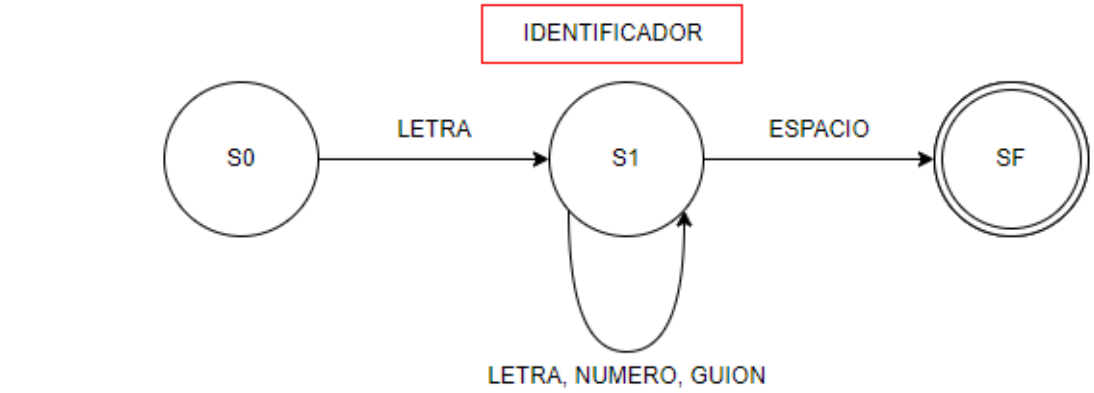


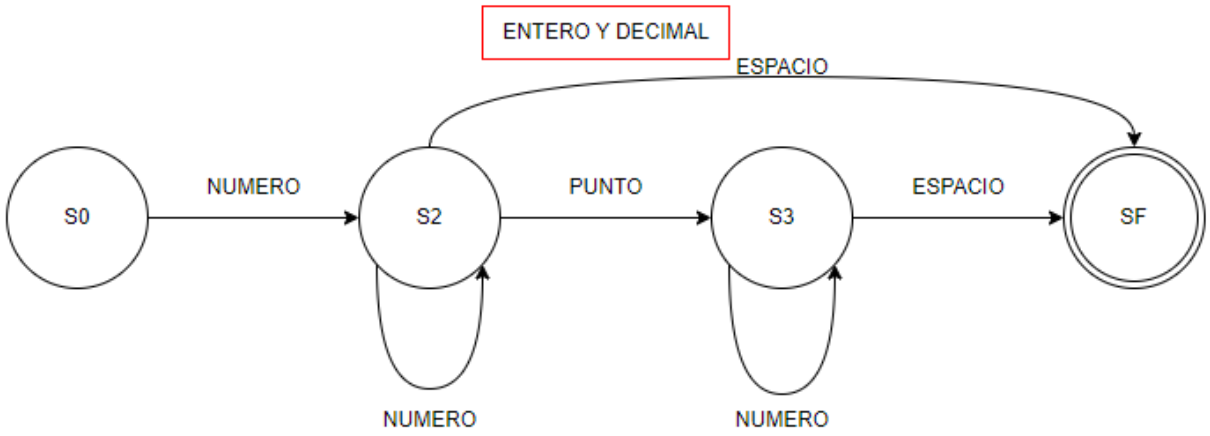
FrontEnd

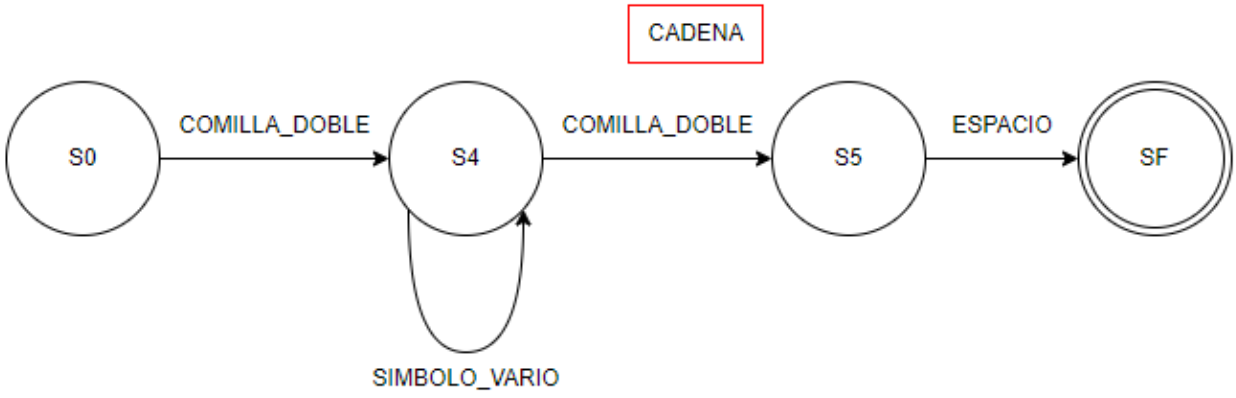
En las siguientes imágenes se detalla cada una de las entidades (clases) usadas para el funcionamiento lógico del aplicativo, en el cual cada una de ellas realiza las funciones posteriormente descritas:

- **InterfazPrincipal:**
    - crearCuadricula: Método que establece el layout que tendrá la cuadricula en donde se pintara el lienzo, establecido el layout también le genera la cuadricula para su posterior pintada.
    - pintarCuadricula: Método que pinta la cuadricula casilla por casilla en base al token que corresponda con su color asignado.
    - iniciarTablero: Método que le da un modelo especifico a la tabla para los reportes para que se puedan mostrar los datos de forma correcta
    - llenarTabla: Método que rellena la tabla para los reportes con los datos del arreglo que contiene todas las tokens que se generaron tras el análisis
    - vaciarTabla: Método que elimina los datos que estén existentes en la tabla para los reportes para que no haya problemas con colapsos de datos

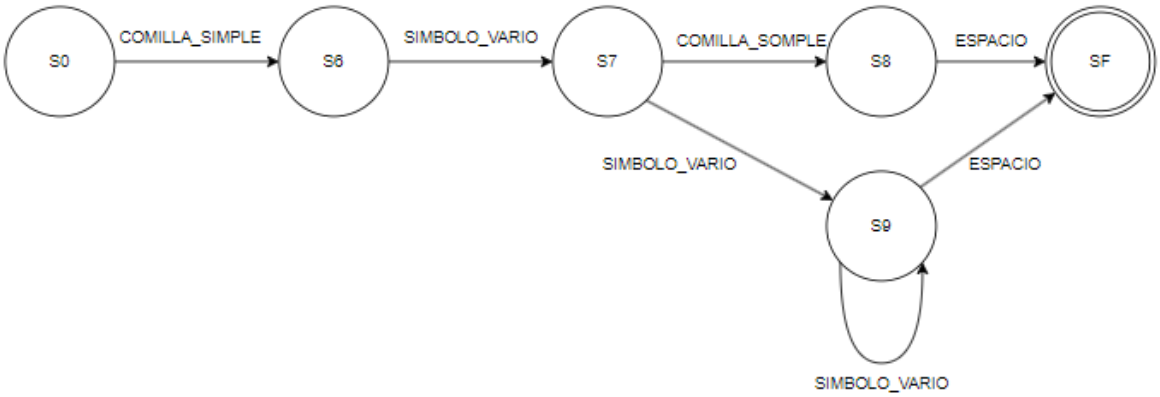
2. AUTOMATAS


IDENTIFICADOR
S0
LETRA
S1
ESPACIO
SF
LETRA, NUMERO, GUION

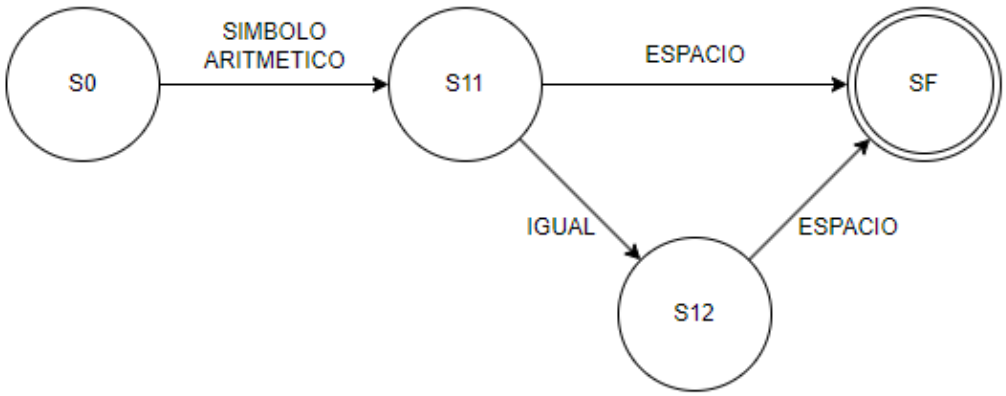

ENTERO Y DECIMAL
ESPACIO
S0
NUMERO
S2
PUNTO
S3
ESPACIO
SF
NUMERO
NUMERO


CADENA
S0
COMILLA_DOBLE
S4
COMILLA_DOBLE
S5
ESPACIO
SF
SIMBOLO_VARIO

CARACTER Y COMENTARIO

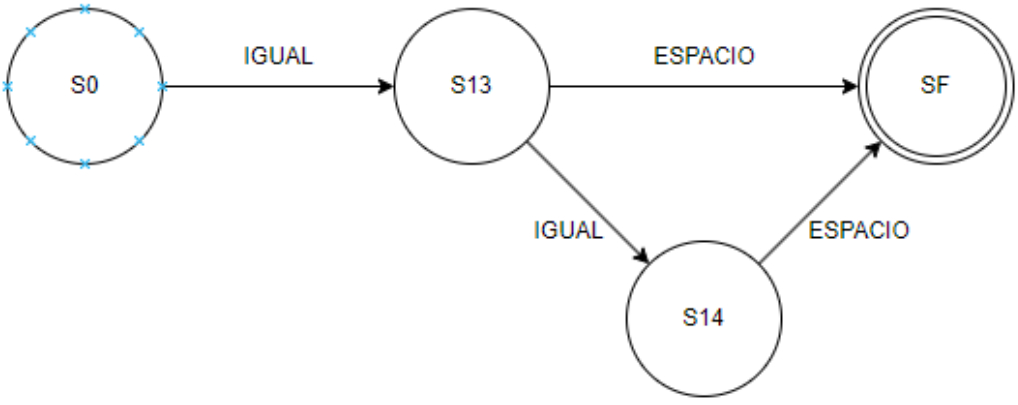


SUMA, RESTA, DIVISION, MULTIPLICACION Y ASIGNACION COMPUESTA

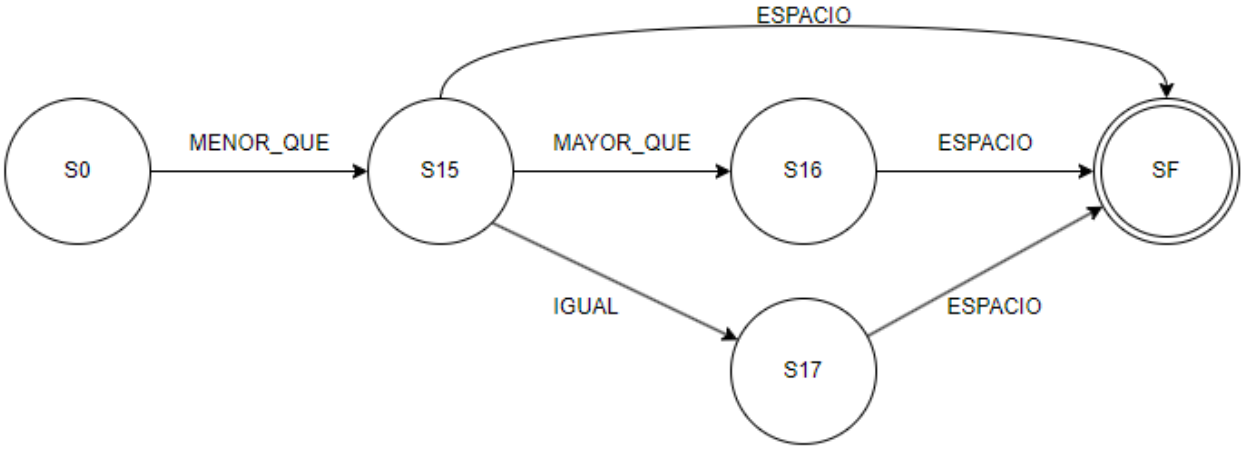


EXPONENTE

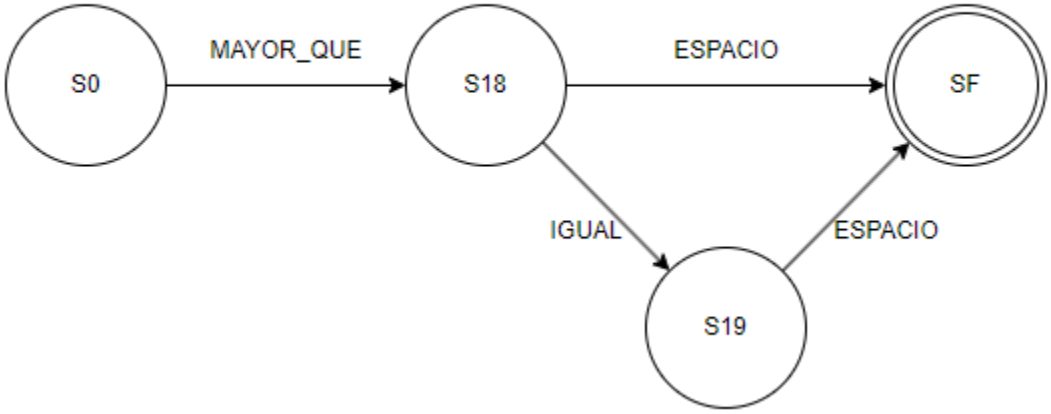
IGUAL Y ASIGNACION SIMPLE



DIFERENTE, MENOR_QUE Y MENOR_IGUAL

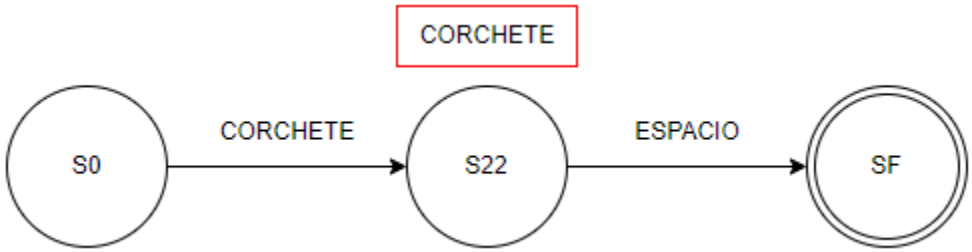


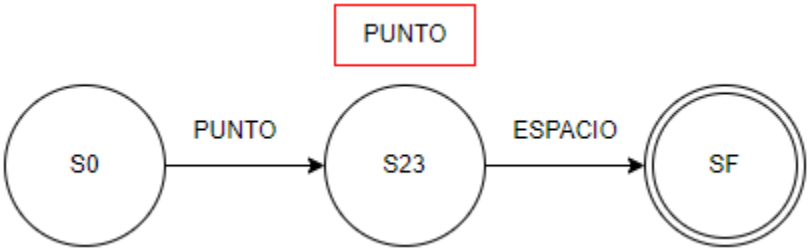
MAYOR_QUE Y MAYOR_IGUAL

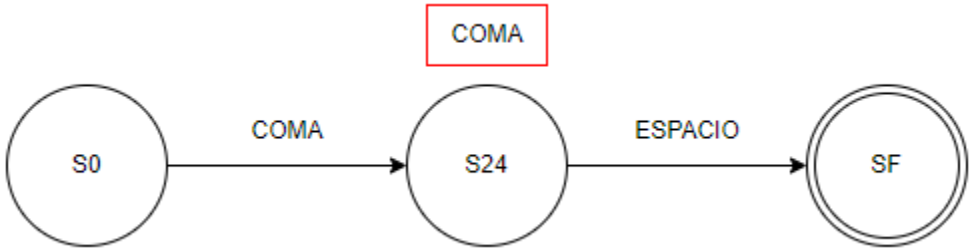


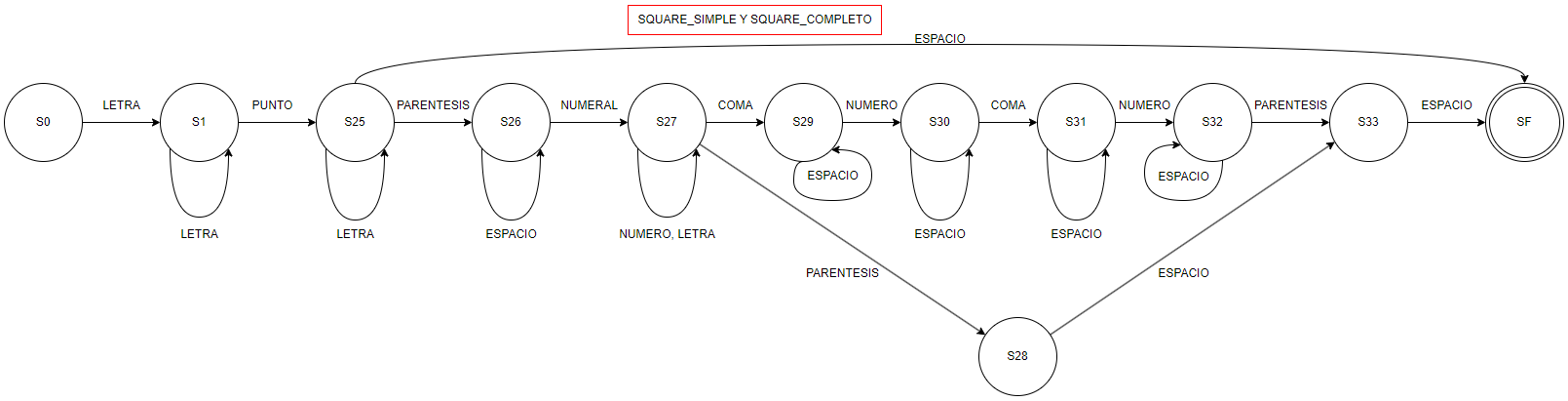
PARENTESIS

LLAVE


CORCHETE
S0
CORCHETE
S22
ESPACIO
SF


PUNTO
S0
PUNTO
S23
ESPACIO
SF


COMA
S0
COMA
S24
ESPACIO
SF


SQUARE_SIMPLE Y SQUARE_COMPLETO
ESPACIO
S0
LETRA
S1
PUNTO
S25
PARENTESIS
S26
NUMERAL
S27
COMA
S29
NUMERO
S30
COMA
S31
NUMERO
S32
PARENTESIS
S33
ESPACIO
SF
LETRA
LETRA
ESPACIO
NUMERO, LETRA
ESPACIO
ESPACIO
ESPACIO
ESPACIO
ESPACIO
PARENTESIS
S28