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.
   1.2. JFlex: Es una herramienta en Java que genera analizadores léxicos, también conocidos como generadores de scaners. Para ello, JFlex toma como entradas una especificación que contiene un conjunto de expresiones regulares y acciones correspondientes.

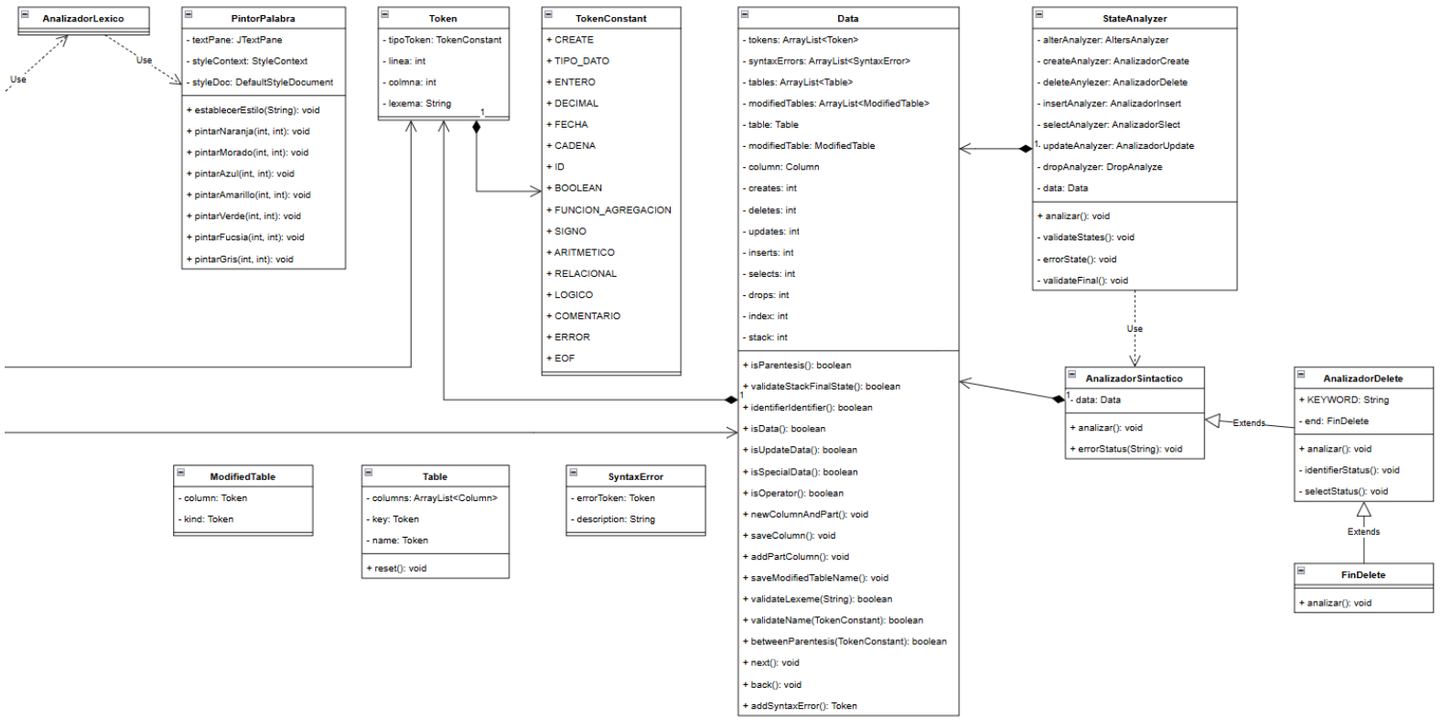
Los analizadores léxicos son programas que leen el código fuente de otro programa y lo transforma en una secuencia de tokens o símbolos. Son la primera fase de un compilador y se encargan de tares como reconocer palabras clase, comentarios y operadores.
JFlex es rápido y no requiere un backtracking costoso. Está diseñado para trabajar con el generador de analizador de LALR CUP, y con la modificación de Berkeley Yacc BYacc/J para Java. También se puede utilizar con otros generadores de analizadores como ANTLR, o como herramientas independientes.

DIAGRAMAS DE MODELAMIENTO

DIAGRAMA DE CLASES

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

1. BACKEND

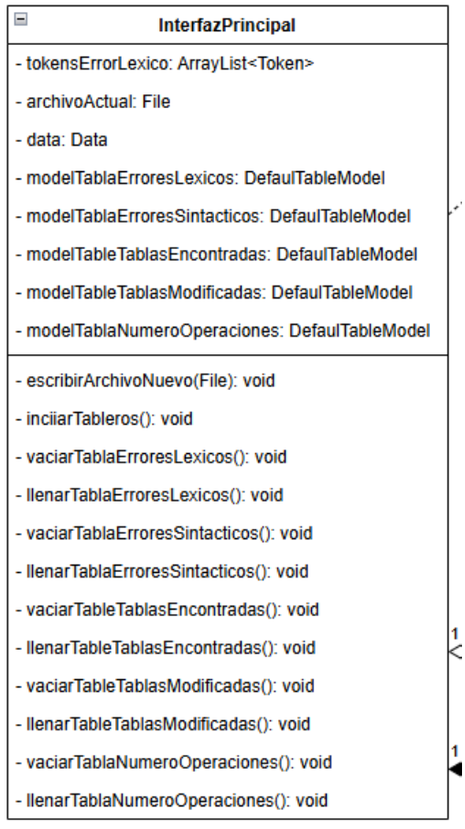


- **AnalizadorLexico:** Clase que genera automáticamente la herramienta de JFlex, por lo tanto no hay mayor explicación.
- **PintorPalabra:** Clase que se encarga de dar estilo al código fuente que se analiza.
    - establecerEstilo(): Método que le da propiedades de estilo al área de texto en donde se ingresa el código fuente de SQL, para que posteriormente se pueda pintar las palabras según los códigos de colores que más adelante se explicaran.
    - pintarNaranja(): Método que establece el color naranja a la palabra que se encuentra en el rango de posición de texto que se le ingresan como parámetro.
    - pintarMorado():Método que establece el color morado a la palabra que se encuentra en el rango de posición de texto que se le ingresan como parámetro.
    - pintarAzul():Método que establece el color azul a la palabra que se encuentra en el rango de posición de texto que se le ingresan como parámetro.
    - pintarAmarillo():Método que establece el color amarillo a la palabra que se encuentra en el rango de posición de texto que se le ingresan como parámetro.
    - pintarVerde():Método que establece el color verde a la palabra que se encuentra en el rango de posición de texto que se le ingresan como parámetro.
    - pintarFucsia():Método que establece el color fucsia a la palabra que se encuentra en el rango de posición de texto que se le ingresan como parámetro.
    - PintarGris():Método que establece el color gris a la palabra que se encuentra en el rango de posición de texto que se le ingresan como parámetro.

- **Token:** Clase que representa a un token el cual será generado por el Analizador Léxico cuando vaya analizando el código fuente de la Aplicación.
- **TokenConstant:** Clase del tipo Enumerado que contiene todos los tipos de token que ir generando el Analizador Léxico.
- **AnalizadorSintactico:** Clase que se encarga de realizar el análisis sintáctico del código fuente realizando el recorrido de los tokens que fueron abstraídos del analizador léxico de JFlex.
    - Analizar(): Método abstracto que es la firma para las clases que hereden de esta, tengan que implementar el análisis en donde cada clase sabrá el código que tendrá que analizar y por ende la distinta implementación que le darán al método.
    - errorStatus(): Método que recoge los errores sintácticos que van apareciendo en el análisis, para irlos agregando a la lista de errores sintácticos que posteriormente servirán para reportes.
- **ModifiedTable:** Clase que se encarga de tener el control de las sentencias que son pertinentes a las modificaciones de tablas, para poder guardar la información de los tokens relacionados que posteriormente servirán para los reportes.
- **SyntaxError:** Clase que se encarga de modelar los errores sintácticos que se van generando durante el análisis del código fuente, en donde captura la información del token y la descripción de lo que se esperaba posterior al token para que no se suscitara el error.
- **Table:** Clase que modela las tablas que se van creando durante el análisis del código fuente, en donde se guarda la información necesaria para que posteriormente se puedan utilizar en los reportes, o para su traficación con Graphviz.
    - Reset(): Método que resetea los atributos de la clase como los son "name" que representa el token identificador de la tabla, "key" representa la clave anterior a la identificación de la tabla y "column" que son las representaciones de las columnas de la tabla.

2. FRONTEND

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:** Clase que se encarga de la vista de la aplicación
    - escribirArchivoNuevo(File): Método que se encarga de escribir el código fuente de la aplicación en un archivo externo para guardar la información.

- iniciarTableros(): Método que le da un modelo especifico a las tablas para los reportes, para que se puedan mostrar los datos de forma correcta
- vaciarTablaErroresLexicos():Método que elimina los datos que estén existentes en la tabla de los reportes de errores léxicos, para que no haya problemas con colapsos de datos
- llenarTablaErroresLexicos(): Método que rellena la tabla para los reportes de errores léxicos con los datos del arreglo que contiene todos los tokens que se generaron tras el análisis y son de error
- vaciarTablaErroresSintacticos():Método que elimina los datos que estén existentes en la tabla de los reportes de errores sintácticos, para que no haya problemas con colapsos de datos
- llenarTablaErroresSintacticos(): Método que rellena la tabla para los reportes de errores sintácticos con los datos del arreglo que contiene los syntaxErrors de la clase Data.
- vaciarTableTablasEncontradas(): Método que elimina los datos que estén existentes en la tabla de los reportes de tablas encontradas, para que no haya problemas con colapsos de datos
- llenarTableTablasEncontradas(): Método que rellena la tabla para los reportes de las tablas encontradas dentro de las sentencias SQL del código fuente ingresado.
- vaciarTableTablasModificadas(): Método que elimina los datos que estén existentes en la tabla de los reportes de tablas modificada, para que no haya problemas con colapsos de datos
- llenarTableTablasModificadas(): Método que rellena la tabla para los reportes de las tablas modificadas dentro de las sentencias SQL del código fuente ingresado.

EXPLICACION DE EXPRESIONES REGULARES

Para las definiciones de las expresiones regulares dentro del archivo .flex, se encuentran expresiones/métodos comunes para todas que se encuentran definidos por JFlex, a continuación se explicaran:

<YYINITIAL>: Es el estado léxico que le indica al analizador léxico que debe de empezar en su estado inicial

Yychar: Nos retorna la posición en la que inicia la cadena encontrada dentro del texto analizado.

Yylenght(): Nos retorna la longitud de la cadena encontrada dentro del texto analizado.

Yyline: Nos retorna el número de línea en la que se encontró la cadena dentro del texto analizado.

Yycolumn: Nos retorna el número de columna en la que se encontró el primer carácter de la cadena dentro del texto analizado.

Yytext(): Nos retorna el contenido de la cadena que coincidió con la expresión regular.

A continuación se explican las expresiones regulares junto con el código que se ejecuta cuando se encuentra la coincidencia de cadena con la expresión regular:

PALABRAS CLAVE DE SQL

- Reservadas: Para las definiciones de las palabras reservadas en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso es la definición literal de la cadena a buscar en el análisis. Seguidamente de instrucciones específicas tales como, si se encuentra una coincidencia entonces se invoca a la función de pintarNaranja() perteneciente a la clase PintorPalabra, que se encargara de pintar la palabra de naranja y luego retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> "CREATE" {pintor.pintarNaranja((int)yychar, yylength()); return new Token(TokenConstant.CREATE, yyline, yycolumn, yytext());}

- Tipos de Dato: Para las definiciones de las palabras reservadas en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso es la definición literal de la cadena a buscar en el análisis. Seguidamente de instrucciones específicas tales como, si se encuentra una coincidencia entonces se invoca a la función de pintarMorado() perteneciente a la clase PintorPalabra, que se encargara de pintar la palabra de morado y luego retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> "INTEGER" {pintor.pintarMorado((int)yychar, yylength()); return new Token(TokenConstant.TIPO_DATO, yyline, yycolumn, yytext());}

- Booleanos: Para las definiciones de las palabras reservadas en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso es la definición literal de la cadena a buscar en el análisis. Seguidamente de instrucciones específicas tales como, si se encuentra una coincidencia entonces se invoca a la función de pintarAzul() perteneciente a la clase PintorPalabra, que se encargara de pintar la palabra de azul y luego retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> "TRUE" {pintor.pintarAzul((int)yychar, yylength()); return new Token(TokenConstant.BOOLEAN, yyline, yycolumn, yytext());}

- Funciones de agregación: Para las definiciones de las palabras reservadas en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso es la definición literal de la cadena a buscar en el análisis. Seguidamente de instrucciones específicas tales como, si se encuentra una coincidencia entonces se invoca a la función de pintarAzul() perteneciente a la clase PintorPalabra, que se encargara de pintar la palabra de azul y luego retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> "SUM" {pintor.pintarAzul((int)yychar, yylength()); return new Token(TokenConstant.FUNCION_AGREGACION, yyline, yycolumn, yytext());}

- Lógicos: Para las definiciones de las palabras reservadas en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso es la definición literal de la cadena a buscar en el análisis. Seguidamente de instrucciones específicas tales como, si se encuentra una coincidencia entonces se invoca a la función de pintarNaranja() perteneciente a la clase PintorPalabra, que se encargara de pintar la palabra de naranja y luego retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> "AND" {pintor.pintarNaranja((int)yychar, yylength()); return new Token(TokenConstant.LOGICO, yyline, yycolumn, yytext());}

IDENTIFICADORES: **[a-z]+([a-z] | _ | [0-9])***

Para las definiciones de los identificadores en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso dara el formato para cadenas **snake_c453**. Seguidamente de instrucciones específicas tales como, si se encuentra una coincidencia entonces se invoca a la función de pintarFucsia() perteneciente a la clase PintorPalabra, que se encargara de pintar la palabra de fucsia y luego retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> {LETRA_MINUSCULA}+({LETRA_MINUSCULA}|[_]|{NUMERO})* {pintor.pintarFucsia((int)yychar, yylength()); return new Token(TokenConstant.ID, yyline, yycolumn, yytext());}

NUMEROS ENTEROS Y DECIMALES: **[0-9]+ [0-9]+ . [0-9]+**

Para las definiciones de los números enteros y decimales en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso dara el formato números enteros y decimales. Seguidamente de instrucciones específicas tales como, si se encuentra una coincidencia entonces se invoca a la función de pintarAzul() perteneciente a la clase PintorPalabra, que se encargara de pintar la palabra de azul y luego retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> {NUMERO}+ {pintor.pintarAzul((int)yychar, yylength()); return new Token(TokenConstant.ENTERO, yyline, yycolumn, yytext());}

<YYINITIAL> {DECIMAL} {pintor.pintarAzul((int)yychar, yylength()); return new Token(TokenConstant.DECIMAL, yyline, yycolumn, yytext());}

FECHAS: '**[0-9][0-9][0-9][0-9] - [0-9][0-9] - [0-9][0-9]'**

Para las definiciones de las fechas en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso dara el formato a las fechas. Seguidamente de instrucciones específicas tales como, si se encuentra una coincidencia entonces se invoca a la función de pintarAzul() perteneciente a la clase PintorPalabra, que se encargara de pintar la palabra de azul y luego retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> """{NUMERO}{NUMERO}{NUMERO}{NUMERO}"-"{NUMERO}{NUMERO}"-"{NUMERO}{NUMERO}""" {pintor.pintarAmarillo((int)yychar, yylength()); return new Token(TokenConstant.FECHA, yyline, yycolumn, yytext());}

CADENAS DE TEXTO

Para las definiciones de las cadenas de texto en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso dara el formato a las cadenas de texto que acepatara cualquiere tipo de carácter dentro de comillas simples. Seguidamente de instrucciones específicas tales como, si se encuentra una coincidencia entonces se invoca a la función de pintarVerde() perteneciente a la clase PintorPalabra, que se encargara de pintar la palabra de verde y luego retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> """([\\] | [^\n\'])*""" {pintor.pintarVerde((int)yychar, yylength()); return new Token(TokenConstant.CADENA, yyline, yycolumn, yytext());}

OPERADORES Y SIGNOS

- Signos: Para las definiciones de los signos en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso es la definición literal de la cadena a buscar en el análisis. Seguidamente de instrucciones específicas tales como retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> "(" {return new Token(TokenConstant.SIGNO, yyline, yycolumn, yytext());}

- Operadores Aritméticos: Para las definiciones de los operadores aritmeticos en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso es la definición literal de la cadena a buscar en el análisis. Seguidamente de instrucciones específicas tales como retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> "+" {return new Token(TokenConstant.ARITMETICO, yyline, yycolumn, yytext());}

- Operadores Relacionales: Para las definiciones de los operadores relacionales en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso es la definición literal de la cadena a buscar en el análisis. Seguidamente de instrucciones específicas tales como retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> "<" {return new Token(TokenConstant.RELACIONAL, yyline, yycolumn, yytext());}

COMENTARIOS DE LINEA

Para las definiciones de los comentarios de linea en el lenguaje SQL se presenta el siguiente ejemplo implementado en el archivo .flex, en donde se encuentra la expresión regular que en este caso dara el formato números enteros y decimales. Seguidamente de instrucciones específicas tales como, si se encuentra una coincidencia entonces se invoca a la función de pintarGris() perteneciente a la clase PintorPalabra, que se encargara de pintar la palabra de gris y luego retorna un objeto del tipo Token que representa a la cadena encontrada

<YYINITIAL> "-"" ""-".* {pintor.pintarGris((int)yychar, yylength()); return new Token(TokenConstant.COMENTARIO, yyline, yycolumn, yytext());}

ESPACIOS Y NUEVAS LINEAS: **[" "\r\t\n\f]**

Para este caso solamente se ignoraran

<YYINITIAL> {ESPACIO} {}

ERRORES

Las siguientes expresiones se utilizan para controlar los errores léxicos, es decir, palabras que no estén definidas dentro del lenguaje o caracteres que igual no estén definidos en el lenguaje:

<YYINITIAL> . {return new Token(TokenConstant.ERROR, yyline, yycolumn, yytext());}