MANUAL TÉCNICO
PROYECTO 1 LENGUAJES FORMALES DE PROGRAMACIÓN

INTRODUCCIÓN

La finalidad de todo manual técnico es la de proporcionar al lector las pautas de configuración y la lógica con la que se ha desarrollado una aplicación, la cual se sabe que es propia de cada programador; por lo que se considera necesario ser documentada.

Aclarando que este manual no pretende ser un curso de aprendizaje de cada una de las herramientas empleadas para el desarrollo del sitio, sino documentar su aplicación en el desarrollo del sitio. Para un mayor detalle acerca de cada una de las herramientas utilizadas, y su forma de operación y aplicación, se recomienda consultar los manuales respectivos de cada una de ellos.

Este manual expondrá o será explicado de manera que uno o varios programadores tengan conocimiento de como fue creada la aplicación y cual es la funcionalidad de cada uno de los métodos y funciones que se encuentran en el código.

```
public static void main(String[] args) {
    FrameAnalizadorLexico frameAnalizadorLexico = new FrameAnalizadorLexico();
    frameAnalizadorLexico.setLocationRelativeTo(null);
    frameAnalizadorLexico.setVisible(true);
}
```

Método main(String[] args): este método no es más que el lugar donde la aplicación iniciará cuando esta se ejecute. Este método lo lleva a un Jframe llamado frameAnalizadorLexico(), el cual tiene como fin mostrar una ventana en la cual el usuario pueda escribir el código ya sea para: html, css, js.

```
public FrameAnalizadorLexico() {
    initComponents();
}
```

Metodo FrameAnalizadorLexico(): este método no es mas que el constructor de la clase, el cual tiene la función de poder inicializar todos los componentes gráficos de la aplicación.

```
private void btnAnalizarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String texto = txtAreaCodigoFuente.getText();
    ControladorTokenEstado tokenEstado = new ControladorTokenEstado(texto, this);
}
```

Método btnAnalizarActionPerformed(ActionEvent evt):
En la parte grafica de la aplicación existe un botón el cual iniciara con el análisis del código escrito en el JtextArea. Este método lo llevara a una clase llamada ControladorTokenEstado, esta clase tendrá la función de poder identificar los tokens estados para poder cambiar de analizador lexico.

```
public ControladorTokenEstado(String texto, FrameAnalizadorLexico frameAnalizadorLexico) {
    this.textoCodigo = texto;
    this.frameAnalizadorLexico = frameAnalizadorLexico;
    this.posicion = 0;
    this.caracterActual = this.textoCodigo.charAt(posicion);
    analizarTokenEstado();
}
```

Método constructor ControladorTokenEstado(String texto, FrameAnalizadorLexico frameAnalizadorLexico): Este método es el constructor de la clase FrameAnalizadorLexico, el cual recibe como parámetro texto, el texto no es más que el código escrito en el JtextArea se extrae todo el texto y se almacenará en una variable llamada textoCodigo.
Este método también tendrá la obligación de poder inicializar la posición el carácter actual en base a la posición actual. Y al final de este método lo llevará a un método el cual sera encargado de poder analizar primeramente el tipo de token estado para posteriormente su respectivo análisis dependiendo de su lenguaje.

```
private void analizarTokenEstado(){
    String tokenEstado;
    while (caracterActual != '\0') {
        if (Character.isWhitespace(caracterActual)) {
            avanzarCaracter();
```

Método analizarTokenEstado(): este método será el encargado de poder identificar el tipo de token estado escrito mientras el carácter actual no sea el último carácter del texto a analizar.
Si el token estado es html, este lo llevara a una clase donde se encuentra la lógica para analizar todo el código html.
Y así mismo con los otros tokens estado, serán dirigidos a sus respectivas clases para ser analizados.

```
private String identificarTokenEstado(){
    StringBuilder texto = new StringBuilder();
    while (caracterActual != ']' && caracterActual != ' ' && caracterActual != '\0') {
        texto.append(caracterActual);
        avanzarCaracter();
    }
    avanzarCaracter();//avanza el caracter ']'
    return texto.toString();
}
```

Función identificarTokenEstado(): esta función será la encargada de poder identificar el texto (tipo de lenguaje) y devolverá el texto o nombre del lenguaje a analizar.

```
public void avanzarCaracter() {
    posicion++;
    if (posicion < textoCodigo.length()) {
        caracterActual = textoCodigo.charAt(posicion);
    } else {
        caracterActual = '\0'; //indicando final del texto
    }
}
```

Este método podrá actualizar la variable de tipo char ‘caracterAcual’ de manera que avance en un lugar la posición del texto a analizar.

```
public int getPosicion() {
    return posicion;
}

public void setPosicion(int posicion) {
    this.posicion = posicion;
}

public char getCaracterActual() {
    return caracterActual;
}

public void setCaracterActual(char caracterActual) {
    this.caracterActual = caracterActual;
}

public ListaEnlazada getListaTokensHTML() {
    return listaTokensHTML;
}

public ListaEnlazada getListaEnlazadaCSS() {
    return listaEnlazadaCSS;
}

public ListaEnlazada getListaEnlazadaJS() {
    return listaEnlazadaJS;
}
```

Métodos getters() y setters().

```
public AnalizadorLexicoHtml(String textoCodigo, int posicion, ControladorTokenEstado controladorTokenEstado)
    this.textoCodigo = textoCodigo;
    this.posicion = posicion;
    this.controladorTokenEstado = controladorTokenEstado;
    this.caracterActual = this.textoCodigo.charAt(posicion);//obteniendo caracter de la cadena en la posicion
}
```

Método constructor de la clase AnalizadorLexicoHtml(String textoCodigo, int posicion, ControladorTokenEstado controladorTokenEstado): Este es el método constructor de la clase AnalizadorLexicoHtml, la cual será la encargada de poder identificar cada uno de las palabras reservadas, etiquetas y datos escritos en la ejecución de la aplicación.
Tiene el deber de inicializar con las variables: textoCodigo, posicion, controladorTokenEstado y con el caracterActual.

```
 * Metodo que permite el analisis del codigo de html
 */
public void analizarCodigoHtml() {
    while (caracterActual != '\0' && !cambiarTokenEstado()) {
        if (Character.isWhitespace(caracterActual)) {
            avanzarCaracter();
        } else if (caracterActual == '<') {
```

Método analizarCodigoHtml(): este método será el encargado de analizar todos los tipos de tokens que se presenten el texto. Esto se hará si y solo si el caracterActual sea diferente de vació o que el carácter a analizar sea diferente de los tokens estado.

```
private void identificarTipoEtiqueta() {
    StringBuilder resultado = new StringBuilder();
    while (caracterActual != '>' && caracterActual != ' ' && caracterActual != '\0') {
        resultado.append(caracterActual);
        avanzarCaracter();
    }
    for (int i = 0; i < etiquetas.length; i++) {
        if (resultado.toString().equals(etiquetas[i])) {
            System.out.println("La etiqueta es: " + etiquetas[i]);
            controladorTokenEstado.getListaTokensHTML().agregarElemento(new Token(traducirEtiqueta(et
            if (resultado.toString().equals("area") || resultado.toString().equals("entrada")) {
                etiquetaDeUnaLinea = true;
            } else {
                etiquetaDeUnaLinea = false;
            }
            if (caracterActual == ' ') {
                controladorTokenEstado.getListaTokensHTML().agregarElemento(new Token(" "));
            }
            break;
        }
    }
}
```

Método identificarTipoEtiqueta(): este método será el encargado de poder identificar una etiqueta, para esto se leerá carácter por carácter de una palabra uniendo estos caracteres en una palabra, esto se hará atraves de un ciclo while que tendrá las condiciones de mientras el caracterActual sea diferente de ‘ ‘, ‘>’ o ‘\0’, si estas condiciones se cumplen entonces se avanzará de carácter y se ira agrupando para que forme una palabra.
Cuando la palabra se halla formado entonces se compará con un arreglo de etiquetas se compara cual es la que coincide con la palabra formada y se crea el token de esa etiqueta.

```
private String traducirEtiqueta(String etiquetaATraducir) {
    String etiquetaTraducida = "";
    switch (etiquetaATraducir) {
        case "prinicipal":
            etiquetaTraducida = "main";
            break;
        case "encabezado":
            etiquetaTraducida = "header";
            break;
        case "navegacion":
            etiquetaTraducida = "nav";
            break;
        case "apartado":
```

Método traducirEtiqueta(String etiquetaATraducir): esta función podrá identificar el tipo de etiqueta que recibe como parámetro y podrá traducirla a su equivalente para poder ser escrita en el archivo html.

```
private void avanzarEspaciosEnBlanco() {
    while (caracterActual == ' ') {
        avanzarCaracter();
    }
}
```

Método avanzarEspaciosEnBlanco(): este método permitirá avanzar si caracterActual sea igual a un espacio en blanco.

```
private void identificarPalabrasReservadas() {
    boolean saltarSignoIgual = true;
    StringBuilder resultado = new StringBuilder();
    avanzarEspaciosEnBlanco();
    if (caracterActual == '=') {
        saltarSignoIgual = false;
    }
```

Método identificarPalabrasReservadas(): este método será el encargado de poder identificar si la palabra a analizar es una palabra reservada del lenguaje html.

```
private void almacenarCadena() {
    StringBuilder resultado = new StringBuilder();
    while (caracterActual != '"' && caracterActual != '\0') {
        resultado.append(caracterActual);
        avanzarCaracter();
    }
    controladorTokenEstado.getListaTokensHTML().agregarElemento(new Token(resultado.toString()));
    controladorTokenEstado.getListaTokensHTML().agregarElemento(new Token(String.valueOf(caracterActual)));
    avanzarCaracter();//avanzar caracter '"'
    avanzarEspaciosEnBlanco();
    if (caracterActual != '>') {
        controladorTokenEstado.getListaTokensHTML().agregarElemento(new Token(" "));
    }
    System.out.println("LA CADENA ES: " + resultado.toString());
    avanzarEspaciosEnBlanco();
}
```

Método que permite almacenar un conjunto de caracteres formados.

```
private void identificarTexto() {
    StringBuilder texto = new StringBuilder();
    while (caracterActual != '<' && caracterActual != '\0') {
        texto.append(caracterActual);
        avanzarCaracter();
    }
    controladorTokenEstado.getListaTokensHTML().agregarElemento(new Token(texto.toString()));
    System.out.println("El texto entre ambas etiquetas (apertura y cierre) es: " + texto.toString());
    avanzarEspaciosEnBlanco();
}
```

Método identificarTexto(): metodo que permite identificar un texto dentro de dos tipos de etiquetas, una etiqueta de apertura y otra de cierre.
Esto se hará mediante un ciclo while el cual formara todo el texto dentro de estas dos etiquetas hasta que el caracterActual sea igual a un signo menor que.
Una vez que se haya encontrado el signo menor que, se dejara de seguir analizando el texto y este texto será almacenado en un nuevo token.

```
private boolean cambiarTokenEstado(){
    if (textoCodigo.startsWith(">>", posicion)) {
        System.out.println("Cambiar token estado");
        return true;
    } else {
        return false;
    }
}
/**
```

Función cambiarTokenEstado(): esta función permitirá identificar cuando el caracterActual sea iguala a dos mayor que, esto indicara que existe un token estado. Devolverá verdadero si esto es cierto.

```
public void avanzarCaracter() {
    posicion++;
    if (posicion < textoCodigo.length()) {
        caracterActual = textoCodigo.charAt(posicion);
    } else {
        caracterActual = '\0'; //indicando final del texto
    }
}
```

Método que permite avanzar de carácter.

```
public AnalizadorLexicoCss(String textoCodigo, int posicion, ControladorTokenEstado controladorTokenEstado) {
    this.textoCodigo = textoCodigo;
    this.posicion = posicion;
    this.controladorTokenEstado = controladorTokenEstado;
    this.caracterActual = this.textoCodigo.charAt(posicion);
}
```

Método constructor AnalizadorLexicoCss(String textoCodigo, int posicion, ControladorTokenEstado controladorTokenEstado): el método constructor es el encargado de poder inicializar con las variables textoCodigo, posicion y controladorTokenEstado.

```
public void analizarCodigoCss() {
    while (caracterActual != '\0' && !cambiarTokenEstado()) {
        if (Character.isWhitespace(caracterActual)) {
            controladorTokenEstado.getListaEnlazadaCSS().agregarElemento(new Token(" "));
            avanzarCaracter();
        } else if (caracterActual == '{') {
```

Método analizarCodigoCss(): este método sera el encargado de recorrer todos los caracteres hasta que se encuentre con un token estado.

En este método se comparara si caracterActual, es igual a uno de los caracteres especiales de ser así se genera un token de no ser así entonces se va a un método el cual sera el encargado de poder identificar el tipo de palabra que es.

```
private void identificarSelectorEtiqueta() {
    boolean buscarSelectorEnReglas = true;
    StringBuilder texto = new StringBuilder();
    while (caracterActual != '\0' && caracterActual != ' ' && caracterAct
            && caracterActual != ';' && caracterActual != '\n') {
        texto.append(caracterActual);
        avanzarCaracter();
    }
```

Método que permite identificar el tipo de selecto (etiqueta) a analizar, al ser un caracater el que se está analizando, se busco la manera de que se pudiera formar una palabra avanzado de carácter hasta que se encuentre un carácter “espacio en blanco”. De esta manera se puede formar la palabra y posteriormente hacer la comparación con el tipo de selector.

```
private void identificarSelectorReglas(String texto) {
    boolean buscarSelectorEnOtros = true;
    for (int i = 0; i < reglas.length; i++) {
        if (texto.equals(reglas[i])) {
            buscarSelectorEnOtros = false;
            controladorTokenEstado.getListaEnlazadaCSS().agregarElemento(
            System.out.println("El selector REGLAS es: " + texto);
            if (caracterActual == ' ') {
```

Método identificarSelectorReglas(String texto): este método será el encargado de poder identificar si la palabra recibida por parámetro es una de las palabras selector de tipo reglas. Esto se puede hacer ya que todos los selectores están almacenados en arreglos, entonces solo se recorre el arreglo y se va comparando con la palabra a analizar.

```
private void identificarSelectorOtros(String texto) {
    boolean buscarSelectorEnIdentificadores = true;
    for (int i = 0; i < otros.length; i++) {
        if (texto.equals(otros[i])) {
            buscarSelectorEnIdentificadores = false;
            controladorTokenEstado.getListaEnlazadaCSS().agregarElemento(new To
            System.out.println("El selector OTROS es: " + texto);
            avanzarCaracter();//avanza el ultimo caracter de los selectores otr
            if (caracterActual == ' ') {
                avanzarCaracter();
                controladorTokenEstado.getListaEnlazadaCSS().agregarElemento(ne
                avanzarEspaciosEnBlanco();
```

Método identificarSelectorOtros(String texto): Este método permite identificar un selector de tipo otros, recibe como parámetro la palabra que se desea identificar o comparar.
A través de un ciclo for se puede ir comparando si la palabra se encuentra en el arreglo de palabras, si la palabra coincide con alguna de las palabras dentro del arreglo este termina su ciclo y crea un nuevo token. En caso de que no este en el arreglo la palabra, esta palabra se trasladara a otro método para, de la misma manera poder compara con otro arreglo de palabras.

```
private void identificadorSelectorClase() {
    StringBuilder texto = new StringBuilder();
    while (posicion < textoCodigo.length() && (Character.isLetterOrDigit(texto
            || textoCodigo.charAt(posicion) == '-' || textoCodigo.charAt(posic
            || textoCodigo.charAt(posicion) != ' ')) {
        texto.append(caracterActual);
        avanzarCaracter();
    }
    Pattern patron = Pattern.compile(EXPRESION_REGULAR_SELECTOR_CLASE);
    Matcher igualador = patron.matcher(texto.toString());
    if (igualador.matches()) {
        controladorTokenEstado.getListaEnlazadaCSS().agregarElemento(new Token
```

Método identificadorSelectorClase(): este método al igual que el interior tratara la manera de identificar o compara la palabra, de manera que cumpla con una expresión regular, si cumple con la expresión regular, este generará un nuevo token con la palabra de la expresión regular.

```
private void identificadorSelectorID() {
    boolean buscarEnColores = true;
    StringBuilder texto = new StringBuilder();
    while (posicion < textoCodigo.length() && caracterActual != ';' && (Charact
            || textoCodigo.charAt(posicion) == '-' || textoCodigo.charAt(posici
            || textoCodigo.charAt(posicion) != ' ')) {
        texto.append(caracterActual);
        avanzarCaracter();
    }
    Pattern patron = Pattern.compile(EXPRESION_REGULAR_SELECTOR_ID);
    Matcher igualador = patron.matcher(texto.toString());
    if (igualador.matches()) {
```

Método identificadorSelectorID(): este método al igual que el anterior tiene la misma funcionalidad, lo que cambia en este es el tipo de expresión regular la cual es:
Inicia siempre con un punto (‘#’). Seguido de una letra minúscula solo acepta letras minúsculas, luego de la primera letra, pueden seguir números o letras, guion medio para separar las palabras de la clase.

```
private void leerCadena() {
    StringBuilder cadena = new StringBuilder();
    avanzarCaracter();//avanza el caracter "'"
    while (caracterActual != '\'' && caracterActual != '\0') {
        cadena.append(caracterActual);
        avanzarCaracter();
    }
    controladorTokenEstado.getListaEnlazadaCSS().agregarElemento(new Token(caden
    System.out.println("La cadena almacenada es: " + cadena.toString());
    if (caracterActual == '\'') {
        controladorTokenEstado.getListaEnlazadaCSS().agregarElemento(new Token("
        avanzarCaracter();//avanza el caracter "'"
    }
```

Método leerCadena(): Este método va a permitir leer cadenas de texto. Este proceso se hace por medio de un ciclo “while”, la condición es que el carácter actual tiene que ser distinto del carácter ‘ y de carácter vacio. Cuando no se cumplan estas condiciones este saldrá del ciclo y creará un nuevo token con el texto extraido.

```
private boolean cambiarTokenEstado() {
    if (textoCodigo.startsWith(">>", posicion)) {
        System.out.println("Cambiar token estado");
        return true;
    } else {
        return false;
    }
}
```

Método cambiarTokenEstado(): este método permite identificar cuando hay un token estado. Esto se hace por medio de un prefijo, si el prefijo existe entonces identifica que es un cambio de token estado.

VERSIONES DE OS PROGRAMAS.

Aquí se mostrará las versiones que se utilizaron para el desarrollo de dicha aplicación.

- Versión JKD: openjdk version "21.0.4"
- Versión Netbeans: IDE 22