

Introducción

El análisis léxico constituye la primera fase para un compilador, aquí se lee el programa fuente de izquierda a derecha y se agrupa en componentes léxicos (tokens), que son secuencias de caracteres que tienen un significado. Además, todos los espacios en blanco, líneas en blanco, comentarios y demás información innecesaria se elimina del programa fuente. También se comprueba que los símbolos del lenguaje (palabras clave, operadores, etc.) se han escrito correctamente.

También podemos agregar que el análisis sintáctico es la segunda fase para un compilador ya que esta analiza que todos sus componentes estén puestos de una forma correcta.

¿Cómo funciona?

Ejecutar el .jar y luego dándole clic en la pestaña de traducir.

¿Quién puede utilizar?

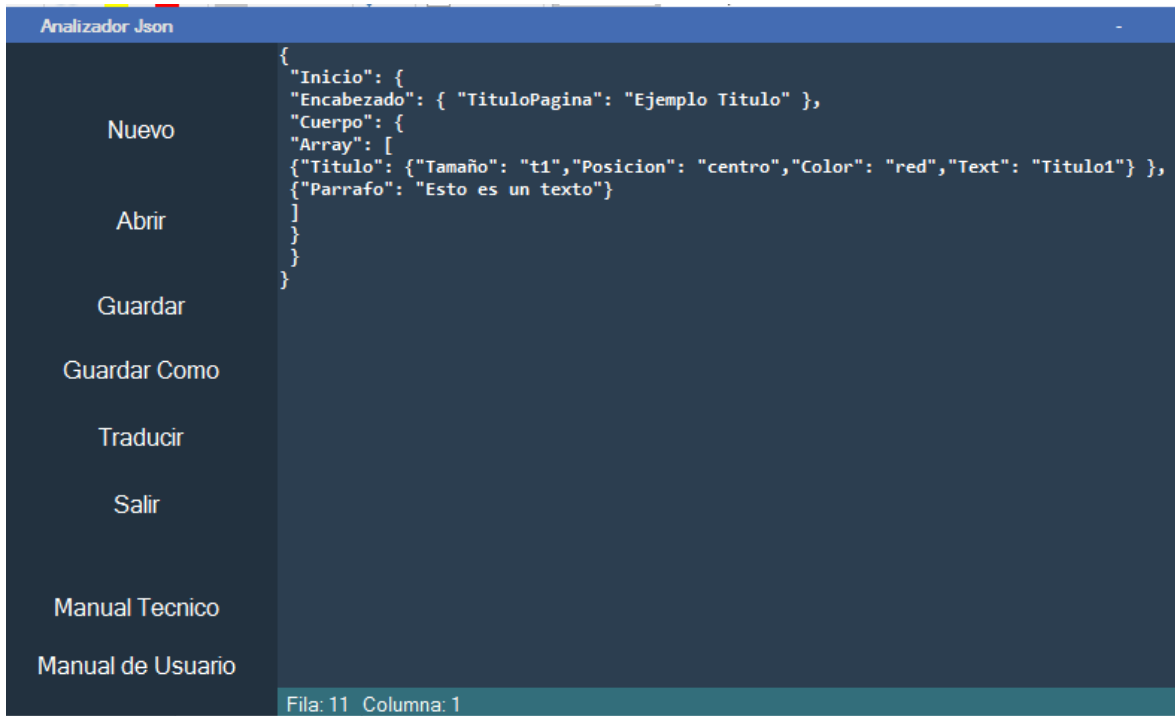
Lo puede utilizar cualquier persona que pueda utilizar una computadora, personas con conocimientos en Compiladores, Autómatas y Lenguajes Formales o herramienta para privados de ingeniería.

Objetivos del sistema:

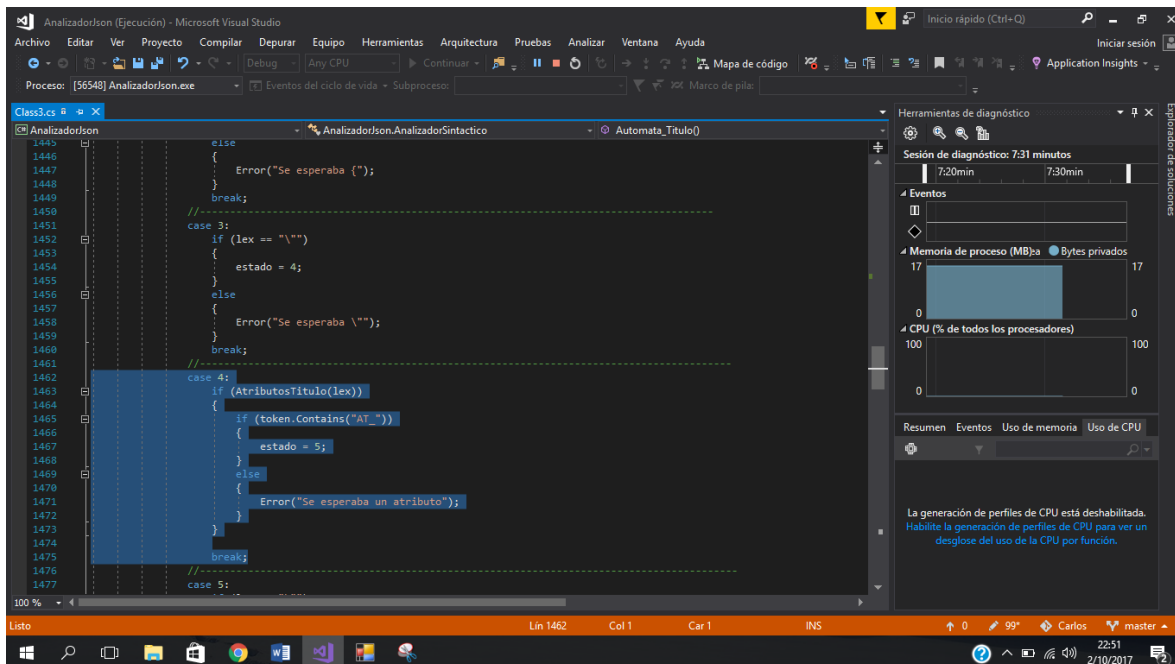
1. Analizar a detalle una instrucción escrita en Pseudocódigo y Codificarla para demostrar su descomposición al usuario.
2. Generar tokens o componentes léxicos de una manera sencilla que nos facilitará la comprensión de la utilidad de un compilador.
3. Generar los errores léxicos y sintácticos de la entrada en resumen poder analizar el texto para que este bien escrito y con una buena estructura.
3. traducir del Lenguaje JSON a HTML.

.

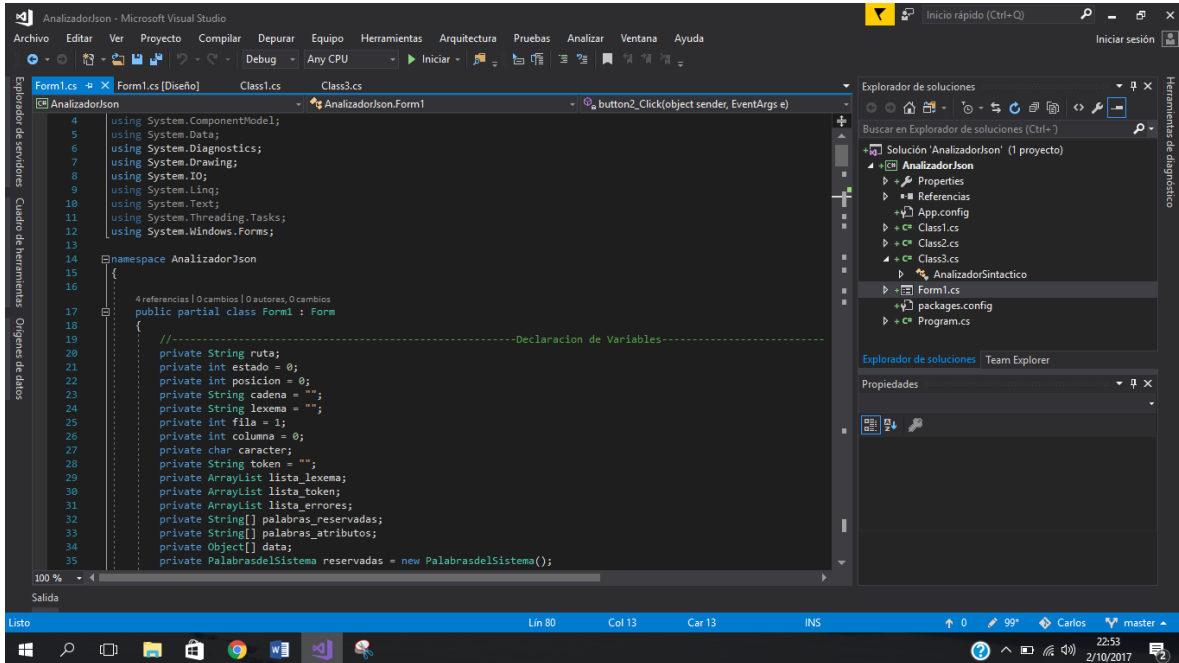
Guía de Software:



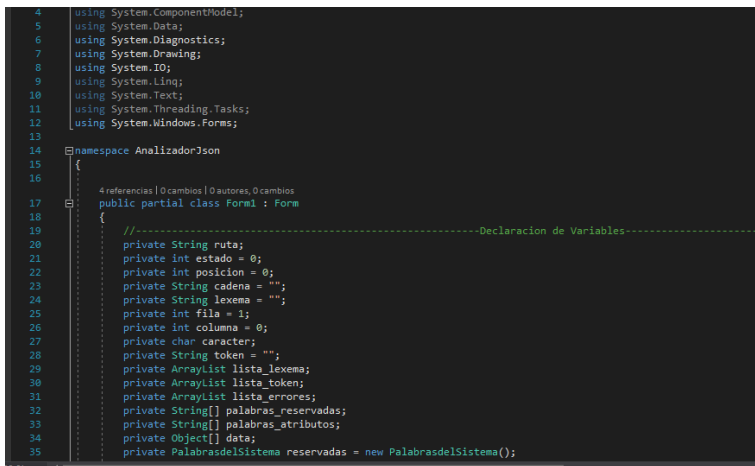
En la captura se puede observar que el diseño de la aplicación fue hecha con la paleta del IDE VISUAL BASIC 2017.



Estas son las librerías que utilizamos para esta aplicación.



Aquí se declararon las variables privadas así solo esa clase en específico tiene acceso a ellas estas variables se declararon privadas ya que se utilizan en toda la clase y se ahorra líneas de código declararlas de esa manera por otro motivo no redundamos en la declaración de variables.



La clase principal llamada AnalizadorLexico es donde está todo el respectivo código del analizador de tokens como pueden ser la palabras reservadas, los atributos, signos de puntuación, errores lexicos y también lo que es texto, esto por medio de un automata ya que se ira analizando con la variable carácter cada letra de dicho texto para poder diferenciar en que estado debe permanecer.

Como se puede observar en estos ifs su objetivo es verificar si el lexema corresponde a un signo de puntuación este corresponde al estado del automata 0 cuando es signo de puntuación, si se encuentra un carácter diferente se transfiere al estado numero 1 para que se analice de otra manera.

```

255 2 //Comentarios:
256 2 referencias | 0 cambios | 0 autores, 0 cambios
257 private void Analizadortexto()
258 {
259     for (int i = 0; i < cadena.Length; i++)
260     {
261         character = cadena[posicion];
262         switch (estado)
263         {
264             case 0: // Para simbolos
265                 if (character == '('')
266                 {
267                     lexema = lexema + Char.ToString(character);
268                     saveArray(lexema, fila, columna);
269                     lista token.Add("Sg_llave_apertura");
270                     lexema = "";
271                 }
272                 else if (character == ')')
273                 {
274                     lexema = lexema + Char.ToString(character);
275                     saveArray(lexema, fila, columna);
276                     lista token.Add("Sg_llave_clausura");
277                     lexema = "";
278                 }
279                 else if (character == ':')
280                 {
281                     lexema = lexema + Char.ToString(character);
282                     saveArray(lexema, fila, columna);
283                     lista token.Add("Sg_dspuntos");
284                     lexema = "";
285                 }
286                 else if (character == '.')
287                 {

```

En este caso:1 se puede observar que indica cuando el autómata cambia de estado de ser un signo de puntuación a ser un carácter de tipo letra o número se va concatenando las letras y números para saber si se obtiene alguna palabra reservada, atributo o simplemente un texto, si en dicho caso el lexema obtiene un signo de puntuación este se regresara al case:0 y así se hará el ciclo hasta terminar de analizar todos los caracteres.

The screenshot shows the Visual Studio Code editor with the following details:

- Top Bar:** Editor, Ver Proyecto, Compilar, Depurar, Equipo, Herramientas, Arquitectura, Pruebas, Analizar, Ventana, Ayuda.
- Activity Bar:** Explorer, Search, Source Control, Run and Debug, Extensions.
- Explorer Panel:**
 - Class2.cs
 - Form1.cs (selected)
 - Class3.cs
- File Explorer:**
 - AnalizadorForm1
 - button2_Click(object sender, EventArgs e)
- Code Editor (Form1.cs):**

```

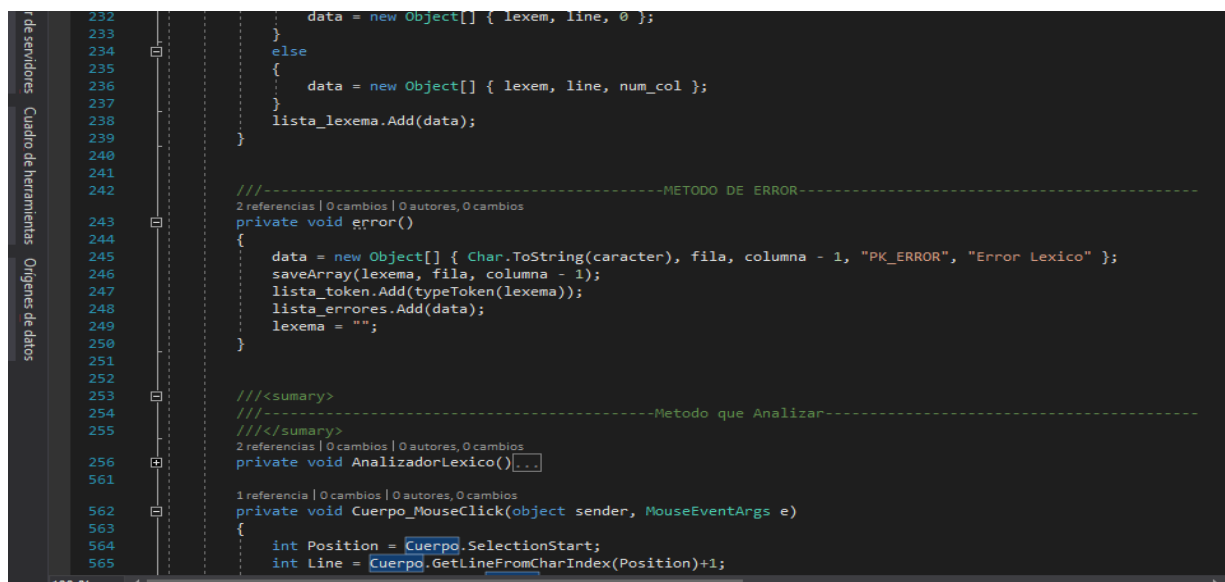
165
166
167
168
169
170
171
172 //-----FUNCIONES-----
173 //-----0 cambios | 0 autores, 0 cambios-----
174 //-----GENERAR HTML DE SALIDA LEXEMAS, TOKENS-----
175 private void generarTablas()
176 {
177     String salida = "";
178     salida += "<center><b><h1>Lista de Lexemas y Tokens</h1><b></center><center><table border=\"1\" style=\"font-weight:bold\"><tr><td>No.</td><td>Lexema</td><td>Tokens</td><td>Fila</td><td>Columna</td></tr><tr><td>1</td><td>lexema</td><td>tokens</td><td>1</td><td>1</td></tr></table></center>";
179     for (int i = 0; i < lista_lexema.Count; i++)
180     {
181         Object[] data = (Object[])lista_lexema[i];
182         salida += "<tr><td>1 + i + "</td><td>";
183     }
184     salida += "</table></center>";
185     TextWriter archivo;
186     archivo = new StreamWriter("TOKENS.html");
187     archivo.WriteLine(salida);
188     archivo.Close();
189
190
191
192
193
194 //-----Verificar que tipo de Token es-----
195 //Referencias | 0 cambios | 0 autores, 0 cambios
196 private String typeToken(String lexema)

```

En el método crearTabla su objetivo principal es que coloque todos los lexemas y tokens encontrados en un archivo .HTML para poder ver una tabla con todos los tokens y lexemas ya que estos nos servirán para poder encontrar los errores si en dicho caso hubieran en la entrada y también nos permite visualizar cuantas palabras reservadas, atributos o textos contiene la entrada Json.

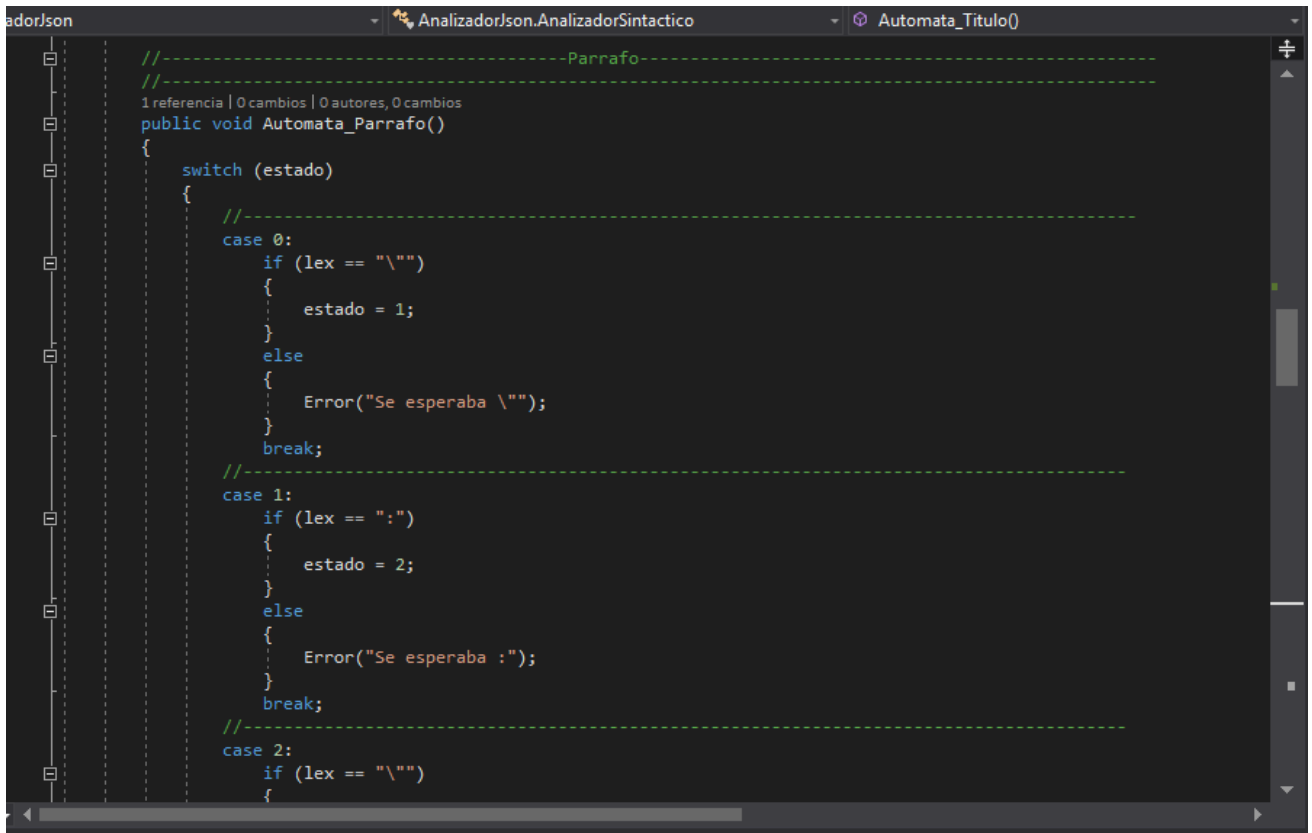
El la variable salida tipo string se utiliza para guardar los valores que necesitamos introducir dentro del archivo HTML y así poder mostrar la salida necesaria.

Metodo main es el que ejecuta la clase principal Principal aca encontraremos las librerías necesarias para la ejecución de la aplicación y dentro del método podemos observar que se llama a la aplicación para poder hacerla visible y que corra sin ningún problema.



```
232     data = new Object[] { lexem, line, 0 };
233 }
234 else
235 {
236     data = new Object[] { lexem, line, num_col };
237 }
238 lista_lexema.Add(data);
239 }
240
241
242
243 //-----METODO DE ERROR-----
244 2 referencias | 0 cambios | 0 autores, 0 cambios
245 private void error()
246 {
247     data = new Object[] { Char.ToString(caracter), fila, columna - 1, "PK_ERROR", "Error Lexico" };
248     saveArray(lexema, fila, columna - 1);
249     lista_token.Add(typeToken(lexema));
250     lista_errores.Add(data);
251     lexema = "";
252 }
253
254 //-----Metodo que Analizar-----
255 //-----summary-----
256 2 referencias | 0 cambios | 0 autores, 0 cambios
257 private void AnalizadorLexico()...
258
259 1 referencia | 0 cambios | 0 autores, 0 cambios
260 private void Cuerpo_MouseClick(object sender, MouseEventArgs e)
261 {
262     int Position = Cuerpo.SelectionStart;
263     int Line = Cuerpo.GetLineFromCharIndex(Position)+1;
```

En este método se encuentran los errores léxicos que pudieran haber en el texto como por ejemplo la @ + - etc. (Caracteres desconocidos). Encontrados los errores se generara un html como salida en el cual se mostraran todos los errores de la entrada Json esto con el fin de tener un mejor analizador.



```
adord/json  AnalizadorJson.AnalizadorSintactico  Automata_Titulo()
//-----Parrafo-----
//-----
1 referencia | 0 cambios | 0 autores, 0 cambios
public void Automata_Parrafo()
{
    switch (estado)
    {
        //-----
        case 0:
            if (lex == "\"")
            {
                estado = 1;
            }
            else
            {
                Error("Se esperaba \"");
            }
            break;
        //-----
        case 1:
            if (lex == ":")
            {
                estado = 2;
            }
            else
            {
                Error("Se esperaba :");
            }
            break;
        //-----
        case 2:
            if (lex == "\"")
            {
                estado = 0;
            }
            else
            {
                Error("Se esperaba \"");
            }
            break;
    }
}
```

En el método se crea un archivo con extensión .HTML con los errores encontrados en el texto este se mostrara en una tabla, Este método Trasladar es el que tiene todo los casos respectivos como por ejemplo si en el texto encuentra la palabra Encabezado este método lo convierte a <head> del lenguaje html

Este método lo traslada todo lo anterior a un archivo .HTML

Es donde se traduce del lenguaje JSON a HTML aquí se encuentran todas las palabras reservadas que puede traer el lenguaje JSON.

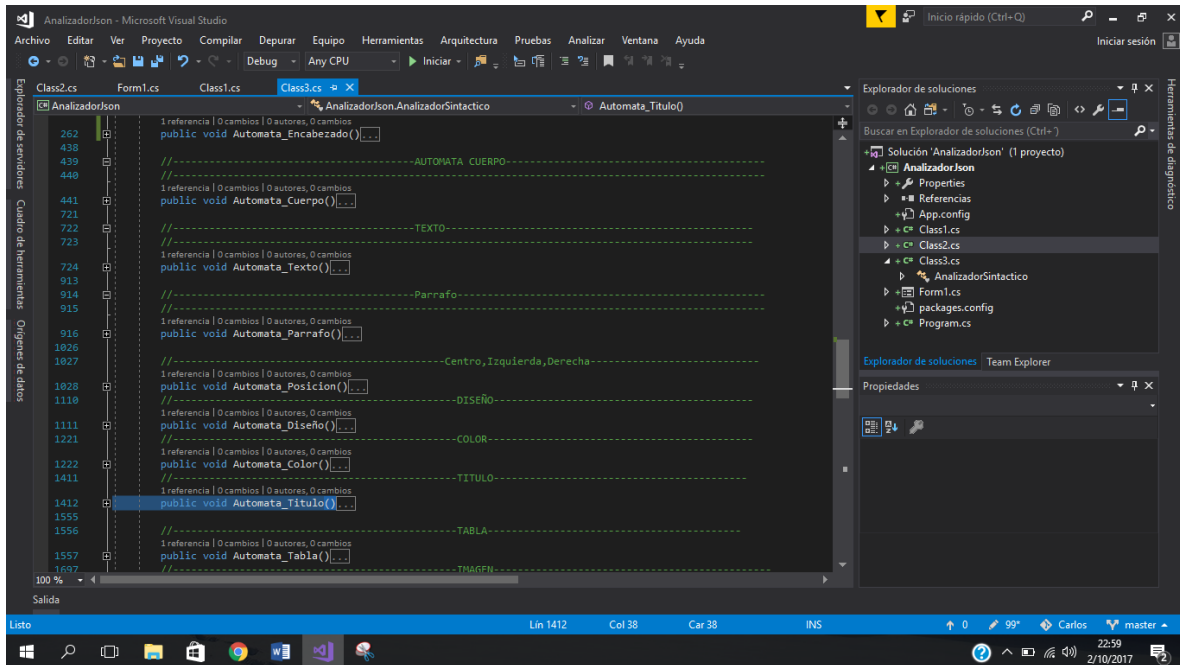
```

}

//-----Metodo para cerrar etiquetas-----
12 referencias | 0 cambios | 0 autores, 0 cambios
private void eliminarEtiquetas()
{
    Boolean veve = false;
    while (veve == false)
    {
        if (lista_etiquetas.Count > 0)
        {
            Object[] etiquet = (Object[])lista_etiquetas[0];
            String ett = (String)etiquet[1];
            html += (String)etiquet[0];
            lista_etiquetas.RemoveAt(0);
            if (ett == "Reservada")
            {
                veve = true;
            }
            if (lista_retornar.Count > 0)
            {
                estado = (int)lista_retornar[0];
                lista_retornar.RemoveAt(0);
            }
            else
            {
                estado = 2;
            }
        }
        else
    }
}

```

En este estado podemos observar que se cierran las etiquetas estas se van cerrando cuando van encontrando una palabra reservada desde la entrada Json.



Este método llamado AuBeg se refiere al automata de la palabra reservada inicio ya que en esta se puede observar que es lo que tiene que tirar si en dicho caso falta o no se encuentra la palabra inicio en el texto Json, si falta alguna llave de apertura o de cierre. Y así se procede de la misma manera con las demás palabras reservadas para el análisis sintáctico.

AUTOMATA GENERAL DE OBJETO

