

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1 A



Manual de Técnico

Wilber Steven Zúñiga Ruano

Carné: 202006629

Guatemala, mayo 2023

Tabla de contenido

Manual de Técnico	1
Introducción	3
Objetivos	3
Requerimientos	3
Manual Técnico	4

Introducción

El presente informe describe los aspectos técnicos de la aplicación con el objetivo de familiarizar al personal técnico con la codificación y funcionalidad.

La codificación de la aplicación se realizó en el lenguaje de JavaScript y sus variantes, se utilizó el paradigma de la Programación Orientada a Objetos para poder encapsular los diversos métodos a utilizar en las diversas clases y estructuras de datos utilizadas.

Objetivos

Describir el diseño, funcionamiento y correcto uso de la aplicación para gestionar su uso, mostrando las especificaciones de los archivos de entrada y partes más relevantes del mismo.

Requerimientos

El sistema puede ser instalado en cualquier versión de Windows que cumpla con los siguientes requerimientos:

- **Nodejs:** v16.13.2
- **Angular CLI:** 13.1.3

Manual Técnico

1. Análisis Léxico y sintactico

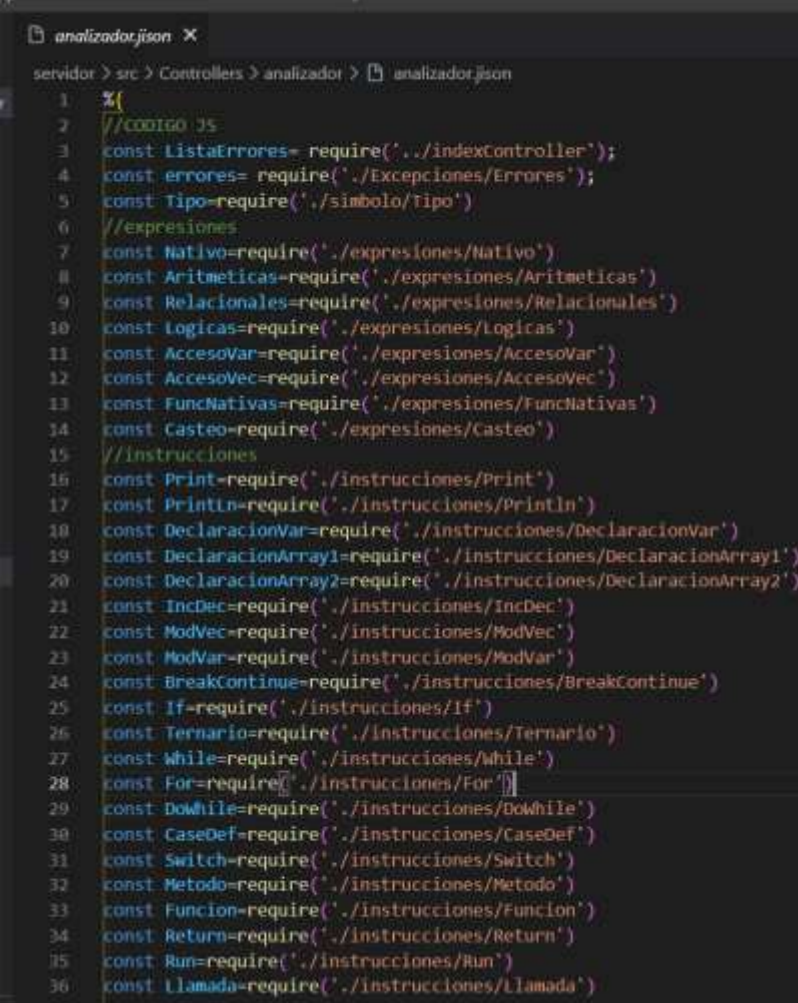
La aplicación realiza un análisis léxico y sintactico con la ayuda de la librería jison para poder reconocer los lexemas de la cadena y convertirlos en tokens que se utilizaran para el análisis sintáctico. Para reconocer los tokens se utilizan expresiones regulares, las cuales definirán que token será cada lexema analizado.

1.1. Crear Analizador léxico con Jison

Para poder crear el analizador léxico con Jison es necesario crear un archivo con extensión jison. Este archivo se divide en segmentos los cuales son:

- Declaración de variables globales
- Declaración de opciones de jison
- Declaración de tokens y expresiones regulares
- Definición de estados

1.2. Ejemplo de archivo jison lexico



```
analizador.jison X
servidor > src > Controllers > analizador > analizador.jison
1  %{
2  //CODIGO JS
3  const ListaErrores= require('../indexController');
4  const errores= require('../Excepciones/Errores');
5  const Tipo=require('../simbolo/Tipo')
6  //expresiones
7  const Nativo=require('../expresiones/Nativo')
8  const Aritmeticas=require('../expresiones/Aritmeticas')
9  const Relacionales=require('../expresiones/Relacionales')
10 const Logicas=require('../expresiones/Logicas')
11 const AccesoVar=require('../expresiones/AccesoVar')
12 const AccesoVec=require('../expresiones/AccesoVec')
13 const FuncNativas=require('../expresiones/FuncNativas')
14 const Casteo=require('../expresiones/Casteo')
15 //instrucciones
16 const Print=require('../instrucciones/Print')
17 const Println=require('../instrucciones/Println')
18 const DeclaracionVar=require('../instrucciones/DeclaracionVar')
19 const DeclaracionArray1=require('../instrucciones/DeclaracionArray1')
20 const DeclaracionArray2=require('../instrucciones/DeclaracionArray2')
21 const IncDec=require('../instrucciones/IncDec')
22 const ModVec=require('../instrucciones/ModVec')
23 const ModVar=require('../instrucciones/ModVar')
24 const BreakContinue=require('../instrucciones/BreakContinue')
25 const If=require('../instrucciones/If')
26 const Ternario=require('../instrucciones/Ternario')
27 const While=require('../instrucciones/While')
28 const For=require('../instrucciones/For')
29 const Dowhile=require('../instrucciones/Dowhile')
30 const CaseDef=require('../instrucciones/CaseDef')
31 const Switch=require('../instrucciones/Switch')
32 const Metodo=require('../instrucciones/Metodo')
33 const Funcion=require('../instrucciones/Funcion')
34 const Return=require('../instrucciones/Return')
35 const Run=require('../instrucciones/Run')
36 const Llamada=require('../instrucciones/Llamada')
```

```
4 %%
5 //palabras reservadas
6 "int"          return 'INT';
7 "double"       return 'DOUBLE';
8 "boolean"      return 'BOOL';
9 "char"         return 'CHAR';
0 "string"       return 'STRING';
1 "new"          return 'NEW';
2 "if"           return 'IF';
3 "else"         return 'ELSE';
4 "switch"       return 'SWITCH';
5 "case"         return 'CASE';
6 "break"        return 'BREAK';
7 "default"      return 'DEFAULT';
8 "while"        return 'WHILE';
9 "for"          return 'FOR';
0 "do"           return 'DO';
1 "continue"     return 'CONTINUE';
2 "return"       return 'RETURN';
3 "print"        return 'PRINT';
4 "println"      return 'PRINTLN';
5 "tolower"      return 'TOLOWER';
6 "toupper"      return 'TOUPPER';
7 "round"        return 'ROUND';
8 "truncate"     return 'TRUNCATE';
9 "length"       return 'LENGTH';
0 "typeof"       return 'TYPEOF';
1 "tostring"     return 'TOSTRING';
2 "tochararray"  return 'TOCHARARRAY';
3 "main"         return 'RUN';
4 "null"         return 'NULL';
5 "true"         return 'TRUE';
6 "false"        return 'FALSE';
7 "void"         return 'VOID';
8
9
0 //comentarios
1 (\/\/.*[^\n]) {}
```

2. Análisis Sintáctico

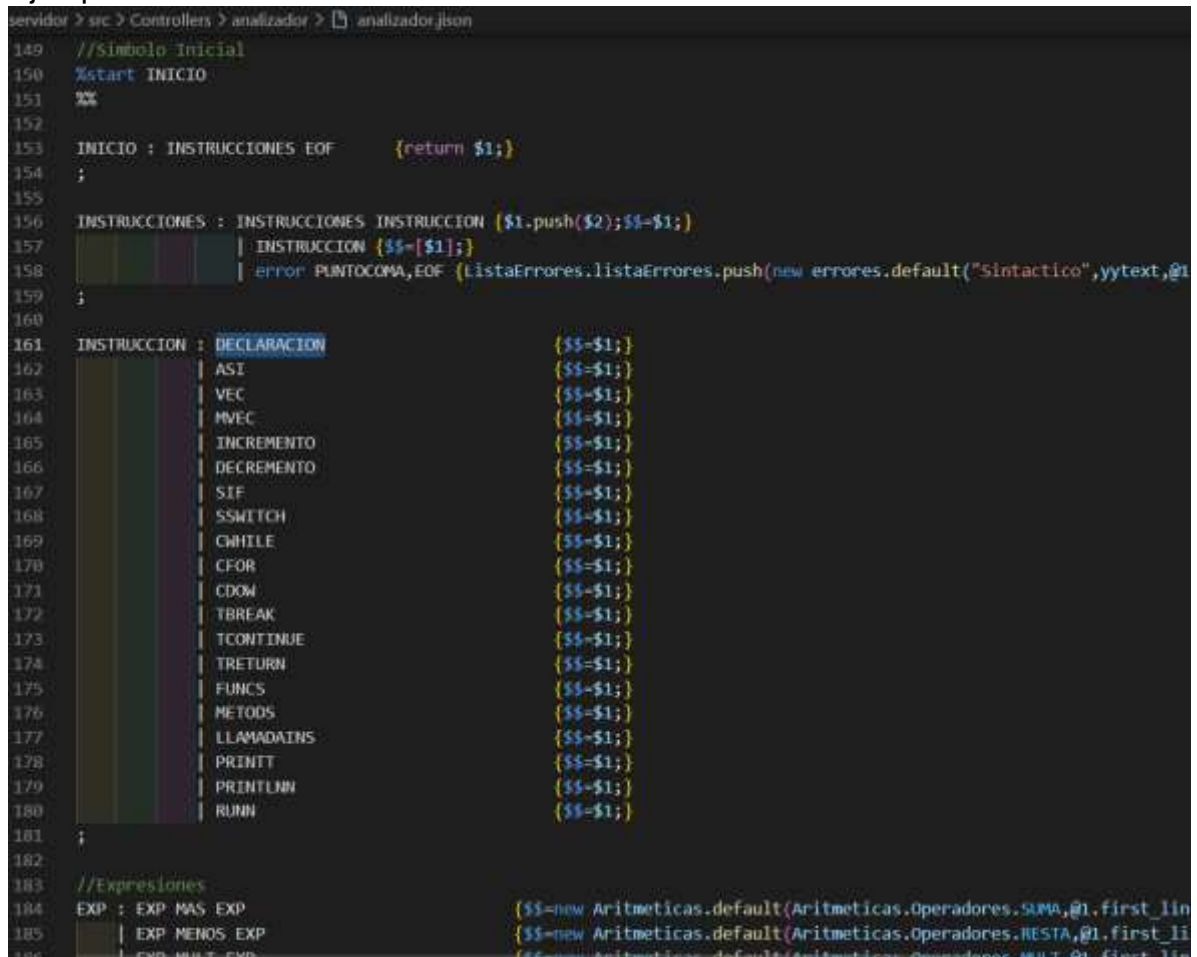
Luego del análisis léxico, la aplicación realiza un análisis sintáctico con la ayuda de la misma librería, la cual permite realizar el análisis sintáctico en base a una gramática.

2.1. Crear el analizador Sintáctico con jison

Para poder crear el analizador sintáctico con jison es necesario crear un archivo con extensión jison. Este archivo se divide en segmentos los cuales son:

- Métodos de errores (recuperables y no recuperables)
- Precedencias
- Declaración de no terminales
- Gramática

2.2. Ejemplo de archivo Jison



```
servidor > src > Controllers > analizador > analizador.jison
149 //Símbolo Inicial
150 %start INICIO
151 %%
152
153 INICIO : INSTRUCCIONES EOF      {return $1;}
154 ;
155
156 INSTRUCCIONES : INSTRUCCIONES INSTRUCCION {$1.push($2);$2=$1;}
157               | INSTRUCCION {$2=$1;}
158               | error PUNTOCOMA,EOF {ListaErrores.listaErrores.push(new errores.default("Sintactico",yytext,@1));}
159 ;
160
161 INSTRUCCION : DECLARACION      {$2=$1;}
162             | ASI              {$2=$1;}
163             | VEC              {$2=$1;}
164             | MVEC             {$2=$1;}
165             | INCREMENTO       {$2=$1;}
166             | DECREMENTO       {$2=$1;}
167             | SIF              {$2=$1;}
168             | SSWITCH          {$2=$1;}
169             | CHWILE           {$2=$1;}
170             | CFOR             {$2=$1;}
171             | CDOW             {$2=$1;}
172             | TBREAK          {$2=$1;}
173             | TCONTINUE       {$2=$1;}
174             | TRETURN         {$2=$1;}
175             | FUNCS            {$2=$1;}
176             | METODS          {$2=$1;}
177             | LLAMADAINS       {$2=$1;}
178             | PRINTT          {$2=$1;}
179             | PRINTLN         {$2=$1;}
180             | RUNN            {$2=$1;}
181 ;
182
183 //Expresiones
184 EXP : EXP MAS EXP              {$2=new Aritmeticas.default(Aritmeticas.Operadores.SUMA,@1.first_line,@1.first_line+1,$1,$2);}
185     | EXP MENOS EXP            {$2=new Aritmeticas.default(Aritmeticas.Operadores.RESTA,@1.first_line,@1.first_line+1,$1,$2);}
186     | EXP MULT EXP             {$2=new Aritmeticas.default(Aritmeticas.Operadores.MULT,@1.first_line,@1.first_line+1,$1,$2);}
```

2.3. Compilación de archivos jison

Una vez terminados el archivo jison es necesario compilarlos con la librería para que estos generen las clases que realizaran el análisis léxico y sintáctico. Para compilarlos se utilizó una clase auxiliar la cual contiene los métodos necesarios para su compilación.

```

public escaneo(req: Request, res: Response) {
  graphQL = "";
  numErrores = 0;
  listaErrores = new Array<Errores>();
  let parser = require("./analizador/analizador");

  try {
    let ast = new Astol(parser.parse(req.body.console));
    let tabla = new TablaSimbolo();
    tabla.setNombre("");
    ast.setTablaGlobal(tabla);
    ast.setConsole("");
    for (let i of ast.getInstrucciones()) {
      if (i instanceof Errores) listaErrores.push(i);
      if (i instanceof Metodo || i instanceof Funcion) {
        i.id = i.id.toLowerCase();
        ast.addFunciones(i);
      }
      if (i instanceof DeclaracionArray1 || i instanceof DeclaracionArray2 || i instanceof
        var resultado = i.interpretar(ast, tabla);
        if (resultado instanceof Errores) {
          listaErrores.push(resultado);
        }
      }
    }
  }
}

```

2.4. Observaciones importantes

Durante el análisis sintáctico, se almacenan las listas para los árboles en listas. Esto con el objetivo de poder capturar la información y poder utilizarla en la realización de los métodos.

```

ast.setNombre("");
ast.setTablaGlobal(tabla);
ast.setConsole("")

```

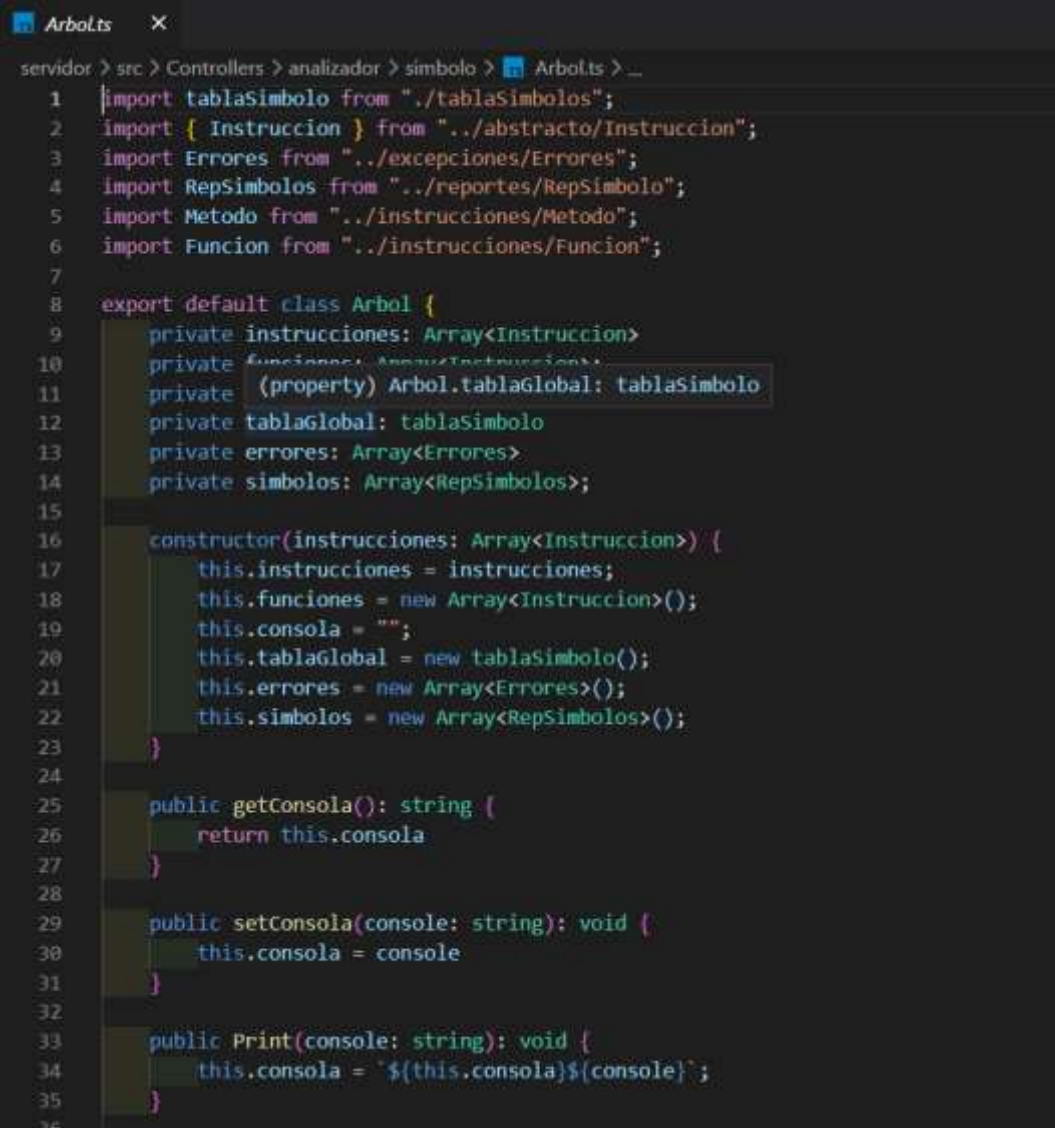
2.5. Método árbol sintáctico:

El proceso de construcción del árbol sintáctico implica analizar el array que utilizamos para identificar los elementos que representan nodos del árbol y las relaciones entre ellos. Luego, se crearían los nodos correspondientes en el árbol y se establecerían las conexiones entre ellos de acuerdo con la estructura del array.

La clase podría tener métodos para agregar nuevos arrays y construir nuevos árboles sintácticos a partir de ellos.

En general, una clase que construye un árbol sintáctico a partir de arrays sería útil para procesar y analizar datos estructurados en un formato específico y representarlos de manera visual y accesible para su manipulación y análisis posterior.

También tiene los returns de cosas que saldrán en la consola



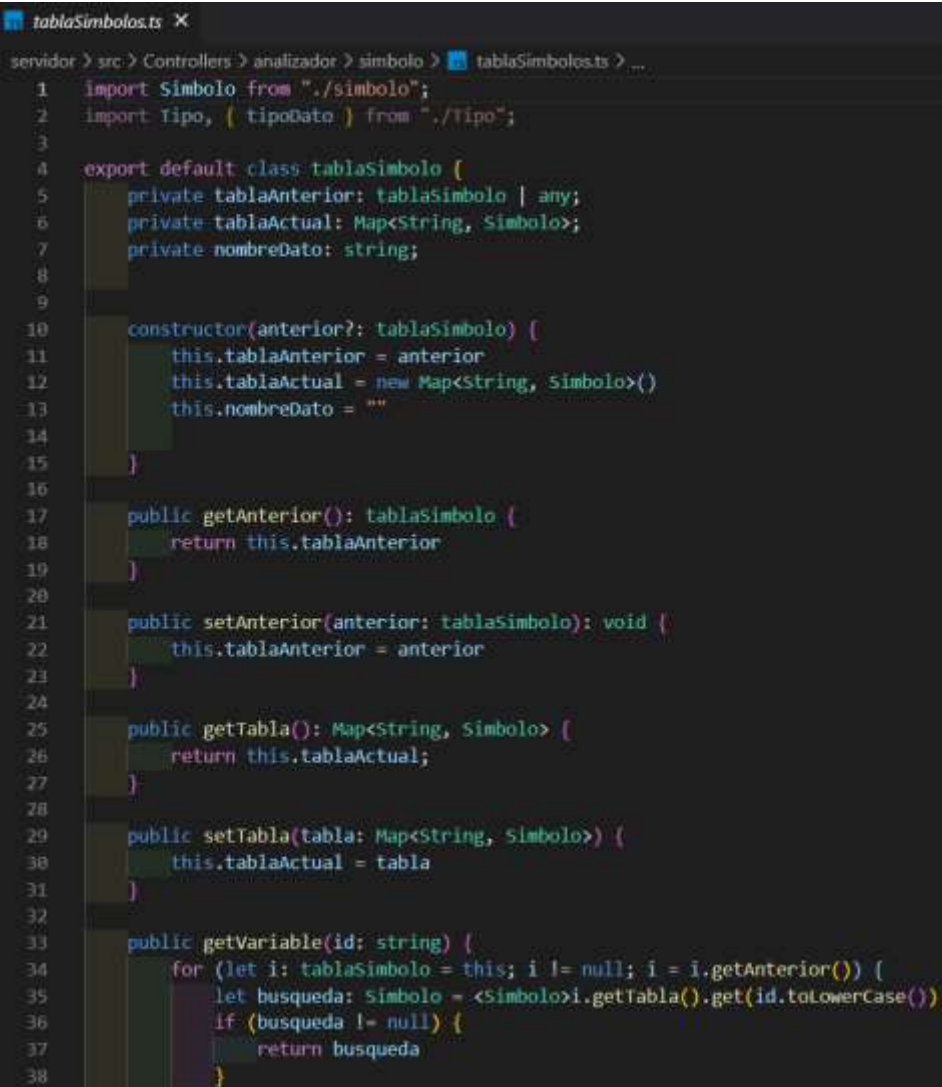
```
Arbol.ts x
servidor > src > Controllers > analizador > simbolo > Arbol.ts > ...
1 import tablaSimbolo from "../tablaSimbolos";
2 import { Instruccion } from "../abstracto/Instruccion";
3 import Errores from "../excepciones/Errores";
4 import RepSimbolos from "../reportes/RepSimbolo";
5 import Metodo from "../instrucciones/Metodo";
6 import Funcion from "../instrucciones/Funcion";
7
8 export default class Arbol {
9     private instrucciones: Array<Instruccion>
10     private funciones: Array<Instruccion>
11     private (property) Arbol.tablaGlobal: tablaSimbolo
12     private tablaGlobal: tablaSimbolo
13     private errores: Array<Errores>
14     private simbolos: Array<RepSimbolos>;
15
16     constructor(instrucciones: Array<Instruccion>) {
17         this.instrucciones = instrucciones;
18         this.funciones = new Array<Instruccion>();
19         this.consola = "";
20         this.tablaGlobal = new tablaSimbolo();
21         this.errores = new Array<Errores>();
22         this.simbolos = new Array<RepSimbolos>();
23     }
24
25     public getConsola(): string {
26         return this.consola
27     }
28
29     public setConsola(console: string): void {
30         this.consola = console
31     }
32
33     public Print(console: string): void {
34         this.consola = `${this.consola}${console}`;
35     }
36 }
```


2.6. Método tabla de símbolos:

La clase que genera una tabla de símbolos sintácticos a partir de otras clases puede ser descrita como una herramienta de análisis estático que escanea el código fuente para identificar los símbolos sintácticos relevantes y crear una tabla que los mapee a sus respectivas definiciones.

La tabla de símbolos resultante proporciona una visión general de la estructura del programa y ayuda a detectar errores y problemas potenciales. Por ejemplo, puede identificar si una variable se ha declarado, pero nunca se usa, o si hay nombres de variables o métodos que no cumplen con las convenciones de nomenclatura adecuadas.

En resumen, la clase que crea una tabla de símbolos sintácticos es una herramienta esencial para la verificación y validación de programas, y ayuda a garantizar que el código cumpla con las convenciones y estándares adecuados.

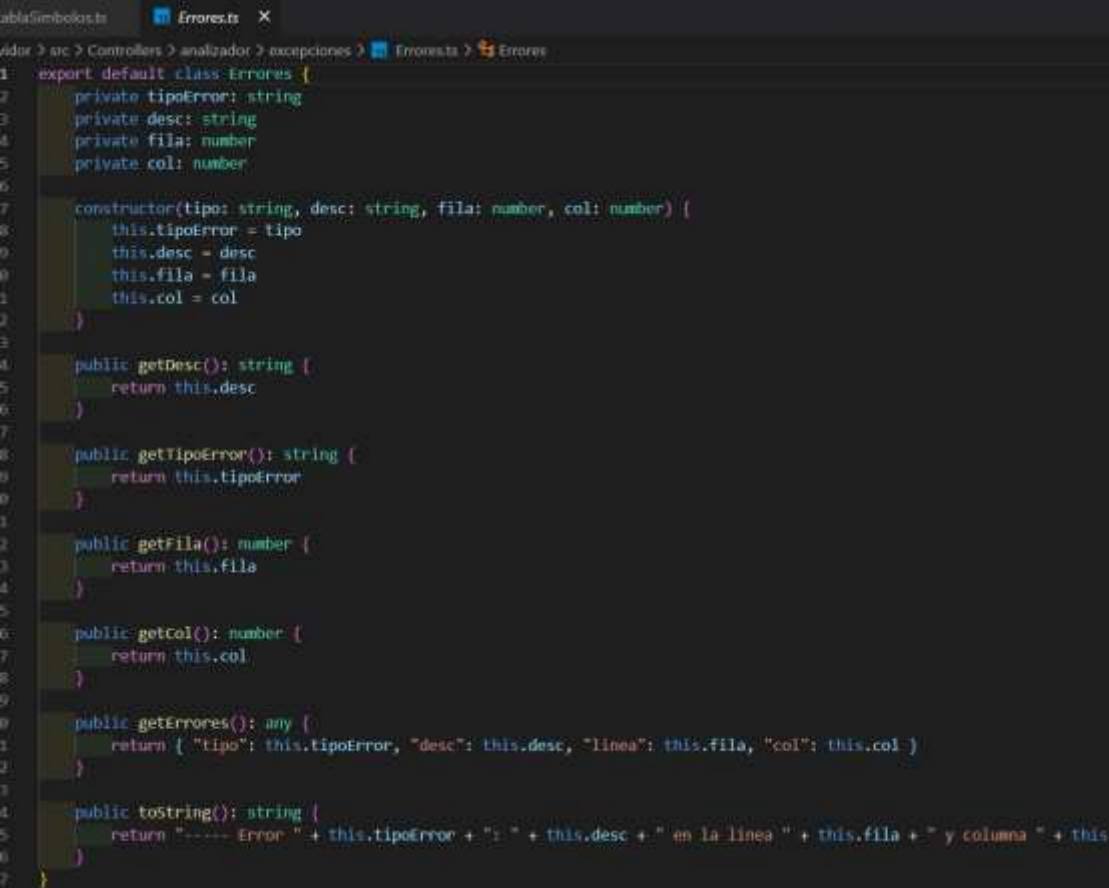


```
1 import Simbolo from "../simbolo";
2 import Tipo, { tipoDato } from "../Tipo";
3
4 export default class tablaSimbolo {
5     private tablaAnterior: tablaSimbolo | any;
6     private tablaActual: Map<String, Simbolo>;
7     private nombreDato: string;
8
9
10    constructor(anterior?: tablaSimbolo) {
11        this.tablaAnterior = anterior
12        this.tablaActual = new Map<String, Simbolo>()
13        this.nombreDato = ""
14    }
15
16
17    public getAnterior(): tablaSimbolo {
18        return this.tablaAnterior
19    }
20
21    public setAnterior(anterior: tablaSimbolo): void {
22        this.tablaAnterior = anterior
23    }
24
25    public getTabla(): Map<String, Simbolo> {
26        return this.tablaActual;
27    }
28
29    public setTabla(tabla: Map<String, Simbolo>) {
30        this.tablaActual = tabla
31    }
32
33    public getVariable(id: string) {
34        for (let i: tablaSimbolo = this; i != null; i = i.getAnterior()) {
35            let busqueda: Simbolo = <Simbolo>i.getTabla().get(id.toLowerCase())
36            if (busqueda != null) {
37                return busqueda
38            }
39        }
40    }
41 }
```

2.7. Método errores:

La clase es una clase que tiene como objetivo generar una tabla de errores a partir de un conjunto de clases y sus respectivos métodos. Esta clase puede ser utilizada en aplicaciones de prueba de software o en sistemas de monitoreo de errores.

Para utilizar esta clase, se deben proporcionar las clases y sus métodos que se desean monitorear. La clase recopila información sobre los errores que ocurren durante la ejecución de los archivos de entrada y genera una tabla que muestra la frecuencia de cada error, así como la clase y el método en el que se produjo.



```
1 export default class Errores {
2     private tipoError: string
3     private desc: string
4     private fila: number
5     private col: number
6
7     constructor(tipo: string, desc: string, fila: number, col: number) {
8         this.tipoError = tipo
9         this.desc = desc
10        this.fila = fila
11        this.col = col
12    }
13
14    public getDesc(): string {
15        return this.desc
16    }
17
18    public getTipoError(): string {
19        return this.tipoError
20    }
21
22    public getFila(): number {
23        return this.fila
24    }
25
26    public getCol(): number {
27        return this.col
28    }
29
30    public getErrores(): any {
31        return { "tipo": this.tipoError, "desc": this.desc, "linea": this.fila, "col": this.col }
32    }
33
34    public toString(): string {
35        return "----- Error " + this.tipoError + ": " + this.desc + " en la linea " + this.fila + " y columna " + this.col
36    }
37 }
```

