

Manual Técnico: Editor de Código en JavaScript

Descripción General

El Editor de Código en JavaScript es una aplicación web desarrollada con JavaScript puro que permite a los usuarios escribir, interpretar y ejecutar código en un lenguaje similar a Java. Este manual técnico proporciona una visión detallada de la arquitectura, componentes y funcionamiento interno del editor, facilitando su mantenimiento, extensión y optimización.

Arquitectura del Sistema

La arquitectura del editor está diseñada siguiendo un enfoque modular, permitiendo una fácil escalabilidad y mantenimiento. Los principales componentes incluyen:

Interfaz de Usuario (UI): Gestiona la interacción con el usuario.

Motor de Interpretación: Procesa y ejecuta el código escrito en el lenguaje similar a Java.

Gestor de Proyectos: Maneja la creación, guardado y carga de proyectos de código.

Componentes Principales

Interfaz de Usuario

La UI está construida con HTML, CSS y JavaScript, proporcionando una experiencia interactiva para el usuario. Los elementos clave incluyen:

Área de Código: Editor de texto para escribir el código.

Botones de Control: Incluyen Ejecutar, Guardar y Nuevo Archivo.

Área de Salida: Muestra la salida del código ejecutado y mensajes de error.

Se uso peggy para el análisis léxico y sintactico del lenguaje



```
1 programa = _ dcl:Declaracion* _ { return dcl }
2
3 Declaracion = dcl:ClassDcl _ { return dcl }
4             / dcl:VarDcl _ { return dcl }
5             / dcl:FuncDcl _ { return dcl }
6             / stmt:Stmt _ { return stmt }
7
8 VarDcl = "var" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, undefined, location()) }
9         / "int" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, Types.INT, location()) }
10        / "float" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, Types.FLOAT, location()) }
11        / "char" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, Types.CHAR, location()) }
12        / "boolean" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, Types.BOOLEAN, location()) }
13        / "string" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, Types.STRING, location()) }
14        / "int" _ id:Identificador _ ";" { return new Declaration(id, null, Types.INT, location()) }
15        / "float" _ id:Identificador _ ";" { return new Declaration(id, null, Types.FLOAT, location()) }
16        / "char" _ id:Identificador _ ";" { return new Declaration(id, null, Types.CHAR, location()) }
17        / "boolean" _ id:Identificador _ ";" { return new Declaration(id, null, Types.BOOLEAN, location()) }
18        / "string" _ id:Identificador _ ";" { return new Declaration(id, null, Types.STRING, location()) }
19
20 FuncDcl = "void" _ id:Identificador _ "(" _ params:Parametros? _ ")" _ bloque:Bloque { return new FunctionDcl(id, params || [], bloque, location()) }
21
22
23 ClassDcl = "class" _ id:Identificador _ "{" _ dcls:ClassBody* _ "}" { return crearNodo('dclClase', { id, dcls }) }
24
25 ClassBody = dcl:VarDcl _ { return dcl }
26            / dcl:FuncDcl _ { return dcl }
27
28 Parametros = id:Identificador _ params:"," _ ids:Identificador { return ids }* { return [id, ...params] }
29
30 Stmt = "System.out.println" _ "(" _ exp:VarlasExpresiones _ ")" _ ";" { return new Print(exp, location()) }
31       / "switch" _ "(" _ exp:Expresion _ ")" _ "{" _ cases:(_ "case" _ exp1:Expresion _ ":" _ stmt:(Stmt)* _ breakForSwitch:("break" _ ";" {return new Break(location())})*) { return new C
32       stmt:(_ "default" _ ":" _ stmt:Stmt(return stmt))? _ "}" { return new Switch(exp, cases, stmt, location()) }
33       / Bloque:Bloque { return Bloque }
34       / "if" _ "(" _ cond:Expresion _ ")" _ "stmtTrue:Stmt
35       stmtFalse:(
36         _ "else" _ stmtFalse:Stmt { return stmtFalse }
37       ){ return new If(cond, stmtTrue, stmtFalse) }
38
39       / "while" _ "(" _ cond:Expresion _ ")" _ "stmt:Stmt { return new While(cond, stmt, location()) }
40       / "for" _ "(" _ init:ForInit _ cond:Expresion _ ";" _ inc:Incremento _ ")" _ "stmt:Stmt {
41       return new For(init, cond, inc, stmt, location()) }
42       }
43       / "break" _ ";" { return new Break(location()) }
44       / "continue" _ ";" { return new Continue(location()) }
45       / "return" _ exp:Expresion? _ ";" { return new ReturnDcl(exp, location()) }
46       / exp:Expresion _ ";" { return new ExpressionStatement(exp, location()) }
47
48
```

Las declaraciones se manejan de esta manera:

```
VarDcl = "var" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, undefined, location()) }
        / "int" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, Types.INT, location()) }
        / "float" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, Types.FLOAT, location()) }
        / "char" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, Types.CHAR, location()) }
        / "boolean" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, Types.BOOLEAN, location()) }
        / "string" _ id:Identificador _ "=" _ exp:Expresion _ ";" { return new Declaration(id, exp, Types.STRING, location()) }
        / "int" _ id:Identificador _ ";" { return new Declaration(id, null, Types.INT, location()) }
        / "float" _ id:Identificador _ ";" { return new Declaration(id, null, Types.FLOAT, location()) }
        / "char" _ id:Identificador _ ";" { return new Declaration(id, null, Types.CHAR, location()) }
        / "boolean" _ id:Identificador _ ";" { return new Declaration(id, null, Types.BOOLEAN, location()) }
        / "string" _ id:Identificador _ ";" { return new Declaration(id, null, Types.STRING, location()) }
```

Permite guardar el tipo en un objeto.

```

Stmnt = "System.out.println" _ "(" _ exp:VariasExpresiones _ ")" _ ";" { return new Print(exp, location()) }
/ "switch" _ "(" _ exp:Expression _ ")" _ "{" _ cases:( _ "case" _ exp1:Expression _ ":" _ stmt:(Stmnt)* _ breakForSwitch:("break" _ ";" {return new Break(location())})* { return
stmt:( _ "default" _ ":" _ stmt:Stmnt(return stmt)? _ ")" { return new Switch(exp, cases, stmt, location()) }
/ Bloque:Bloque { return Bloque }
/ "if" _ "(" _ cond:Expression _ ")" _ "stmtTrue:Stmnt
stmtFalse:(
_ "else" _ stmtFalse:Stmnt { return stmtFalse }
)? { return new If(cond, stmtTrue, stmtFalse) }

/ "while" _ "(" _ cond:Expression _ ")" _ stmt:Stmnt { return new While(cond, stmt, location()) }
/ "for" _ "(" _ "init:ForInit _ cond:Expression _ ";" _ inc:Incremento _ ")" _ stmt:Stmnt {
return new For(init, cond, inc, stmt, location())
}
/ "break" _ ";" { return new Break(location()) }
/ "continue" _ ";" { return new Continue(location()) }
/ "return" _ exp:Expression? _ ";" { return new ReturnDcl(exp, location()) }
/ exp:Expression _ ";" { return new ExpressionStatement(exp, location()) }

```

Tenemos los statements como los for, while, etc. Estos están acá porque su sintaxis es diferente a la de una expresión.

```

Expresion = Asignacion

Asignacion = id:Identificador _ "=" _ asgn:Asignacion { return new Assignment(id, asgn, location()) }
/ id:Identificador _ "+" _ "=" _ asgn:Asignacion { return new Assignment(id, new BinaryOperation(new ReferenceVariable(id, location()), asgn, "+", location()), location()) }
/ id:Identificador _ "-" _ "=" _ asgn:Asignacion { return new Assignment(id, new BinaryOperation(new ReferenceVariable(id, location()), asgn, "-", location()), location()) }
/ exp1:AndOr _ "?" _ exp2:AndOr _ ":" _ exp3:AndOr { return new TernaryOperation(exp1, exp2, exp3, location()) }
/ "parseInt" _ "(" _ exp:Expression _ ")" { return new ParseIntDcl(exp, location()) }
/ "parseFloat" _ "(" _ exp:Expression _ ")" { return new ParseFloatDcl(exp, location()) }
/ "toString" _ "(" _ exp:Expression _ ")" { return new ToStringDcl(exp, location()) }
/ "toLowerCase" _ "(" _ exp:Expression _ ")" { return new ToLowerCaseDcl(exp, location()) }
/ "toUpperCase" _ "(" _ exp:Expression _ ")" { return new ToUpperCaseDcl(exp, location()) }
/ "typeof" _ "(" _ exp:Expression _ ")" { return new TypeOfDcl(exp, location()) }
/ AndOr

AndOr = izq:Comparacion expansion:(
_ op:("&&" / "||") _ der:Comparacion { return { tipo: op, der } }
)* {
return expansion.reduce(
(operationAnterior, operacionActual) => {
const { tipo, der } = operacionActual;
return new BinaryOperation(operationAnterior, der, tipo, location());
},
izq
);
}

Comparacion = izq:MayorMenor expansion:(
_ op:("<=" / ">=") _ der:MayorMenor { return { tipo: op, der } }
)* {
return expansion.reduce(
(operationAnterior, operacionActual) => {
const { tipo, der } = operacionActual;
return new BinaryOperation(operationAnterior, der, tipo, location());
},
izq
);
}

```

Para manejar precedencias usamos expansiones y el de mayor precedencia lo ponemos hasta abajo.

```

Argumentos = arg:Expresion _ args:("," _ exp:Expresion { return exp })* { return [arg, ...args] }

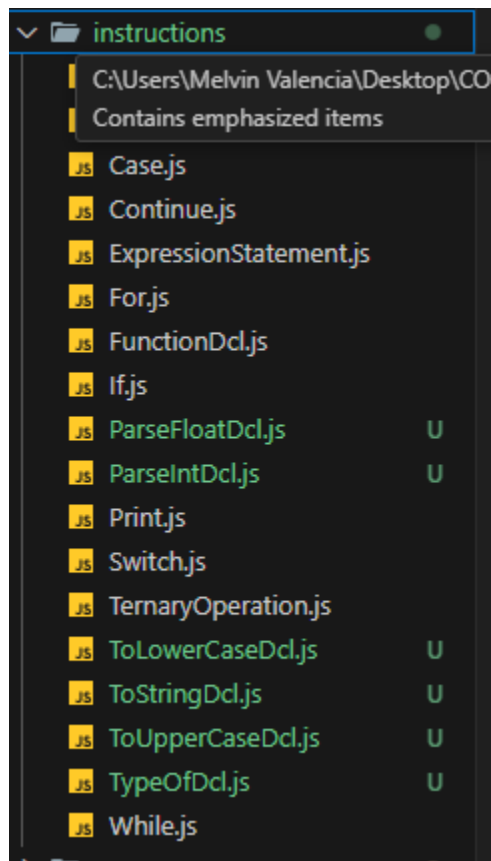
Numero = "(" _ exp:AndOr _ ")" { return new Agrupation(exp, location()) }
/ "[" _ exp:Expresion _ "]" { return new Agrupation(exp, location()) }
/ "[0-9]+( "." [0-9]+ )?" { return new Number(parseFloat(text()), 10), location()) }
/ "true" { return new Boolean(true, location()) }
/ "false" { return new Boolean(false, location()) }
/ "\"\" txt:StringTxt "\"\" { return new String(txt, location()) }
/ "'" txt:normalChar "'" { return new String(txt, location()) }
/ id:Identificador { return new ReferenceVariable(id, location()) }

StringTxt = ["^"]* { return text() }
normalChar = ["^"] { return text() }
_ = ([ \t\n\r] / Comentarios)*

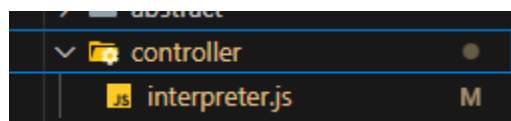
Comentarios = "/*" (!["\n"] .)*
/ "/*" (!["*/"] .)* "*/"

```

De esta manera organizamos algunas expresiones comunes en nuestro lenguaje.



Nuestras clases instrucciones representan las instrucciones en el lenguaje.



Tenemos la clase `interpret` que es donde están nuestros métodos que visitan a nuestras clases.

```
Environment.js M X
src > tools > Environment.js > Environment > saveVariable

1  export default class Environment{
2    constructor(previus=undefined){
3      this.previus = previus;
4      this.tabla = new Map();
5    }
6
7    saveVariable(id, type, value, tree, node){
8      if(this.tabla.has(id)){
9        tree.addError("Variable " + id + " ya existe en el ambiente " + this.id, node.location.start.line, node.location.start.column);
10       return;
11     }
12   this.tabla.set(id, {type, value, linea: node.location.start.line, columna: node.location.start.column});
13 }
14
15   getVariable(id, tree, node){
16     let env = this;
17     while(env != undefined){
18       if(env.tabla.has(id)){
19         return env.tabla.get(id).value;
20       }
21       env = env.previus;
22     }
23     tree.addError("Variable " + id + " no existe en el ambiente " + this.id, node.location.start.line, node.location.start.column);
24     return undefined;
25   }
26
27   getAllVariable(id, tree, node){
28     let env = this;
29     while(env != undefined){
30       if(env.tabla.has(id)){
31         return env.tabla.get(id);
32       }
33       env = env.previus;
34     }
35     tree.addError("Variable " + id + " no existe en el ambiente " + this.id, node.location.start.line, node.location.start.column);
36     return undefined;
37   }
38
39   updateVariable(id, value, tree, node){
40     let env = this;
41     while(env != undefined){
42       if(env.tabla.has(id)){
43         let contenido = env.tabla.get(id);
44         contenido.value = value;
45         env.tabla.set(id, contenido);
46         return;
47       }
48     }
49     tree.addError("Variable " + id + " no existe en el ambiente " + this.id, node.location.start.line, node.location.start.column);
50     return undefined;
51   }
52 }
```

Nuestra clase `environment` que es aquí donde esta nuestra tabla de símbolos.

```

1 import Interpreter from "../controller/Interpreter.js";
2 import {parse} from "../parser/parser.js";
3
4 require.config({ paths: { 'vs': 'https://cdnjs.cloudflare.com/ajax/libs/monaco-editor/0.34.0/min/vs' } });
5 var errorReport = [];
6 var tabla = new Map();
7 require(['vs/editor/editor.main'], () => {
8     const editor = monaco.editor.create(document.getElementById('container'), {
9         language: 'java',
10        theme: 'vs-dark'
11    });
12
13    document.getElementById('run-button').addEventListener('click', () => {
14        const code = editor.getValue();
15        const sentencias = parse(code);
16        console.log(sentencias);
17        const interprete = new Interpreter();
18        sentencias.forEach(sentencia => sentencia.accept(interprete));
19        console.log(interprete.environment.tabla);
20        console.log("Errores: ");
21        console.log(interprete.errors);
22        errorReport = interprete.errors;
23        tabla = interprete.environment.tabla;
24        document.getElementById('output').innerText = interprete.console;
25    });
26
27    document.getElementById('open-file-button').addEventListener('click', () => {
28        const input = document.createElement('input');
29        input.type = 'file';
30        input.accept = '.oak';
31        input.onchange = e => {
32            const file = e.target.files[0];
33            const reader = new FileReader();
34            reader.onload = event => {
35                editor.setValue(event.target.result);
36            };
37            reader.readAsText(file);
38        };
39        input.click();
40    });
41
42    document.getElementById('save-file-button').addEventListener('click', () => {
43        const code = editor.getValue();
44        const blob = new Blob([code], { type: 'text/plain' });
45        const url = window.URL.createObjectURL(blob);
46        const a = document.createElement('a');
47        a.href = url;

```

En el script manejamos el botón run y los botones consecuentes del editor.