

Manual Técnico del Proyecto OakLand

1. Descripción General

Este proyecto, **OakLand**, es un intérprete basado en un lenguaje de programación personalizado, implementado utilizando JavaScript puro. La interfaz gráfica permite escribir, cargar y ejecutar archivos “.oak”, además de generar reportes de errores y de la tabla de símbolos.

2. Tecnologías Utilizadas

- **JavaScript Puro:** El núcleo del proyecto está desarrollado completamente en JavaScript, sin el uso de frameworks o bibliotecas adicionales. Se utilizaron funciones nativas de JavaScript para gestionar la lógica del lenguaje, así como para interactuar con el DOM (Document Object Model) de la interfaz.
- **PEG.js (Parser Expression Grammar):** Se utilizó **PEG.js** para la creación del **parser**. PEG.js es una herramienta que permite generar analizadores sintácticos a partir de gramáticas formales. En este caso, la gramática definida permite procesar las instrucciones del lenguaje OakLand, convertirlas en árboles de sintaxis abstractos (ASTs), y entregarlas al intérprete para su ejecución.
- **Patrón Visitor:** Para el manejo del procesamiento de las instrucciones, se implementó el patrón de diseño **Visitor**, que permite ejecutar comportamientos sobre nodos del AST sin modificar la estructura de dichos nodos. Este patrón fue clave para gestionar:
 - **Variables:** Declaración, asignación, y manejo de tipos.
 - **Arreglos:** Inicialización, manipulación, y acceso a sus elementos.
 - **Sentencias:** Condicionales, bucles, y expresiones.
 - **Clases y Objetos:** Definición de clases, creación de instancias, y manejo de atributos y métodos.

3. Estructura del Proyecto

El proyecto está dividido en varios componentes clave:

- **Parser:** El archivo de gramática generado por **PEG.js** transforma el código fuente .oak en un AST. Este árbol es procesado por el intérprete a través de visitas a cada nodo, aplicando la lógica definida en el patrón **Visitor**.
- **Interprete:** Implementado en JavaScript, este componente recibe el AST del parser y lo recorre utilizando el patrón Visitor. Aquí es donde se gestionan las declaraciones, asignaciones, y ejecución de las distintas instrucciones del lenguaje.
- **Interfaz Gráfica (HTML/CSS):** La interfaz fue desarrollada usando HTML y CSS, ofreciendo una experiencia visual simple pero funcional. El diseño fue optimizado para la entrada y salida de comandos, la carga de archivos, y la generación de reportes. CSS se usó para dar estilo a los botones, áreas de texto y consola, manteniendo una estética minimalista.

4. Detalles de Implementación


- **Manejo de Variables y Tipos:** Las variables pueden ser de diferentes tipos, como `int`, `float`, `string`, `boolean`, y `char`. Las variables se manejan dentro de un entorno dinámico que valida los tipos y sugiere errores cuando se intenta asignar un tipo incorrecto a una variable.
- **Arreglos:** Se implementaron arreglos con soporte para múltiples tipos. El sistema valida que los elementos dentro del arreglo sean consistentes con el tipo declarado, permitiendo manipular datos como listas o colecciones ordenadas.
- **Clases y Objetos:** El proyecto soporta la definición de clases con atributos y métodos, siguiendo el paradigma orientado a objetos. Los objetos creados a partir de clases pueden interactuar mediante invocaciones de métodos y acceso a atributos.
- **Sentencias de Control:** Se implementaron las sentencias condicionales (`if`, `else`) y los ciclos (`while`, `for`), permitiendo la ejecución dinámica de bloques de código en función de ciertas condiciones o iteraciones.


5. Despliegue

El proyecto está diseñado para ser desplegado utilizando **GitHub Pages**. Esto permite que la aplicación web sea accesible desde cualquier navegador sin necesidad de instalación local. El código fuente, incluyendo el parser, el intérprete, y la interfaz, está completamente alojado en un repositorio de GitHub.

6. Conclusión

El proyecto **OakLand** combina el poder de **JavaScript puro** con la flexibilidad de **PEG.js** y el **patrón Visitor** para ofrecer una experiencia completa de interpretación de código a través de una interfaz web. El despliegue en **GitHub Pages** asegura que el proyecto sea fácilmente accesible y mantenible.

FileEditSelectionViewGoRun



EXPLORER

...

▼ PROYECTO 1

▼ entorno

JS entorno.js

▼ parser

JS config.js

JS parser.js

PEG parser.pegjs

▼ patron

▼ funciones

JS foranea.js

JS invocable.js

JS nativas.js

▼ sentencias_trans...

JS transferencia.js

▼ structs

JS instancia.js

JS struct.js

JS interprete.js

JS nodos.js

JS visitor.js

interfaz.css

interfaz.html

JS main.js

JS tool.js

interfa

1 .

3

4

5 }

6

7 ▼ t

8

9

10

11

12

13

14

15

16 }

17

18 /

19 ▼ b

20

21

22

23

24

25 }

26

27 ▼ i

28

29

30 }

```

herazess / input
.container {
  width: 600px;
  margin: 0 auto;
  text-align: center;
}

textarea {
  width: 600px;
  height: 400px;
  margin-bottom: 20px;
  background-color: black;
  color: skyblue;
  border: 2px solid rgb(246, 255, 0);
  border-radius: 5px;
  font-size: large;
}

/* AJUSTE DE SIZE DE AMBOS BOTONES */
button {
  font-size: 18px; /* Aumenta el tamaño del texto dentro del botón */
  padding: 10px 20px; /* Añade relleno para hacer que el botón sea más grande */
  width: 150px; /* Establece un ancho fijo */
  border: 3px solid rgb(0, 217, 255);
  cursor: pointer; /* Cambia el cursor a una mano cuando se pasa por encima */
}

input {
  font-size: 18px;
  width: 170px;
}

/*=====*/

.botones{
  text-align: left;

```

```

export class InterpreterVisitor extends BaseVisitor{

  constructor(){
    super()
    this.entornoActual = new Entorno()

    //funciones nativas
    Object.entries(nativas).forEach(([nombre, funcion]) => {
      this.entornoActual.set(nombre, funcion)
    })

    this.salida = ""
  }

  //ESTOS COSOS DE AQUI ES PARA QUE LOS node DE LOS visit NO PIERDAN EL TIPADO
  /**
   * @type {BaseVisitor['visitOperacionBinaria']}
   */

  visitOperacionBinaria(node){
    const izq = node.izq.accept(this)
    const der = node.der.accept(this)

    switch(node.op){

      case "+":
        return izq + der
      case "-":
        return izq - der
      case "*":
        return izq * der
      case "/":

```

```

import { Invocable } from "../patron/funciones/invocable.js"
import { Instancia } from "../patron/nodos.js"
import { Struct } from "../patron/structs/struct.js"

export class Entorno {
  /**
   * @param {Entorno} padre
   */
  constructor(padre = undefined) {
    this.valores = {} // HashMap
    this.padre = padre
  }

  /**
   En este caso se cambio a nombres simples como set, get y asignar porque ya no se guardan solo variables
   si no tambien funciones, clases y luego arreglos
   */

  /**
   * @param {string} nombre
   * @param {any} valor
   * @param {string} tipo
   */
  set(nombre, { tipo, valor }) {
    //console.log(tipo + "en set de entorno")
    if (valor !== null) {
      if (!this.validacionTipo(tipo, valor, nombre)) {
        this.valores[nombre] = { tipo, valor:null }
        return
      }
      //throw new Error("Error de tipo: Se esperaba un valor de tipo " + tipo + " para la variable " + nombre);
    }
  }
}

```

```

1 <!DOCTYPE html>
2 <html lang="en">
3
4   <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Oakland</title>
8     <link rel="stylesheet" href="interfaz.css">
9   </head>
10
11   <body>
12
13
14     <div class = "botones">
15       <button>Reportes</button>
16       <br></br>
17       <button id="btnCrear">Crear Archivo</button>
18       <br></br>
19       <button id="btnGuardar">Guardar Archivo</button>
20       <br></br>
21       <input type="file" id="fileInput" accept=".oak" />
22     </div>
23     <div class="container">
24       <h2 class ="titulo_oakland">Oakland</h2>
25
26       <textarea id= "editor" placeholder="Escribe tus comandos aquí..."></textarea>
27       <button id="btnEjecutar">Ejecutar</button>
28       <div class="output">
29         <h3>Consola</h3>
30         <textarea id="salida" readonly></textarea>
31       </div>
32     </div>
33     <script type="module" src="main.js"></script>
34
35   </body>
36

```

```

mainjs > ...
1 //const parser = require("../parser/parser.js")
2 import { parse } from "../parser/parser.js"
3 import { InterpreterVisitor } from "../patron/interprete.js"
4
5
6 const editor = document.getElementById("editor")
7 const boton = document.getElementById("btnEjecutar")
8 const consola = document.getElementById("salida")
9 const btnCrear = document.getElementById("btnCrear");
10 const btnGuardar = document.getElementById("btnGuardar");
11
12 boton.addEventListener('click', () => {
13     const codigo_analizar = editor.value
14     console.log(editor.value)
15
16     try {
17         const resultados = parse(codigo_analizar)
18
19         const interprete = new InterpreterVisitor()
20
21         for (const resultado of resultados){
22             resultado.accept(interprete)
23         }
24
25         consola.innerHTML = interprete.salida
26     } catch (error) {
27         //manejo de errores sintacticos
28         console.log(error)
29         consola.innerHTML = error.message + "en la linea: " + error.location.start.line + " y columna: " + error.location.start.column
30     }
31 })
32
33
34
35

```

DIEGO CHEN 202202882