UNIVERSIDAD SAN CARLOS DE GUATEMALA
CENTRO UNIVERSITARIO DE OCCIDENTE
DIVISIÓN DE CIENCIAS DE LA INGENIERÍA

ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1
LABORATORIO

POYECTO FINAL

ESTUDIANTE:
JORGE ANIBAL BRAVO RODRÍGUEZ

CARNÉ:
202131782

AUXILIAR:
ADOLFO SON

Herramientas utilizadas

Java
Java es un lenguaje de programación de propósito general, orientado a objetos, que es ampliamente utilizado para desarrollar aplicaciones de software. Su característica principal es la portabilidad, lo que significa que el código Java puede ejecutarse en cualquier dispositivo que tenga una máquina virtual Java (JVM).

Visual Studio Code
Visual Studio Code es un editor de código fuente desarrollado por Microsoft. Es ligero pero potente, y soporta una amplia gama de lenguajes de programación gracias a su sistema de extensiones.

Apache NetBeans
Apache NetBeans es un entorno de desarrollo integrado (IDE) gratuito y de código abierto que soporta el desarrollo de aplicaciones en Java, PHP, JavaScript y otros lenguajes. Ofrece herramientas para la edición de código, depuración y gestión de proyectos.

Definición gramatical BNF
BNF (Backus-Naur Form) es una notación formal utilizada para describir la sintaxis de lenguajes de programación. Proporciona una forma de definir reglas gramaticales que especifican cómo se estructuran las sentencias en un lenguaje.

Tomcat
Apache Tomcat es un servidor web y contenedor de servlets que se utiliza para ejecutar aplicaciones web Java. Es ampliamente utilizado para implementar aplicaciones web basadas en Java en un entorno de producción.

Ultramarine Linux
Ultramarine Linux es una distribución de Linux basada en Fedora, diseñada específicamente para ser fácil de usar y facilitar la transición de usuarios que provienen de otros sistemas operativos, como Windows. Para los desarrolladores, Ultramarine Linux ofrece un entorno pragmático con varias configuraciones preaplicadas que optimizan la instalación inicial y el uso diario, permitiendo que los usuarios se concentren en su trabajo sin complicaciones adicionales.
Angular: Framework de desarrollo frontend basado en TypeScript, mantenido por Google, que implementa el patrón MVVM (Model-View-ViewModel) y utiliza componentes web para construir aplicaciones SPA (Single Page Applications).

HTML
Lenguaje de marcado estándar utilizado para estructurar y presentar contenido en la web. Define la semántica y estructura de documentos web mediante elementos y atributos organizados jerárquicamente, los cuales son interpretados por navegadores para renderizar páginas web. Es la base fundamental del contenido web y actualmente se encuentra en su versión HTML5.

JavaScript
Lenguaje de programación interpretado, orientado a objetos, débilmente tipado y dinámico, que sigue el estándar ECMAScript. Es un lenguaje multiparadigma que soporta programación funcional, orientada a objetos y basada en prototipos, siendo el lenguaje principal para la programación del lado del cliente en navegadores web, aunque también se usa en el servidor a través de Node.js.

TypeScript
TypeScript es un superset de JavaScript que agrega tipado estático opcional, lo que permite detectar errores de tipo en tiempo de desarrollo. Desarrollado por Microsoft, compila a JavaScript estándar para ejecutar en cualquier entorno compatible con JS. TypeScript mejora la mantenibilidad y escalabilidad de proyectos grandes al ofrecer características como interfaces, tipos avanzados y soporte para programación orientada a objetos.

Node.js
Entorno de ejecución de JavaScript multiplataforma basado en el motor V8 de Chrome, que permite ejecutar código JavaScript del lado del servidor de manera asíncrona y orientada a eventos, con un sistema de módulos y un gestor de paquetes (npm).

Jison
Generador de analizadores sintácticos (parser generator) para JavaScript, basado en la sintaxis de Bison/Yacc, que permite crear parsers ascendentes LALR(1) y LL(k) a partir de una gramática formal.

Firefox
Navegador web de código abierto multiplataforma desarrollado por Mozilla Foundation, que implementa los estándares web actuales (HTML5, CSS3, ECMAScript) y utiliza el motor de renderizado Gecko junto con el motor JavaScript SpiderMonkey.

Servidor web Python
Es utilizado para servir las páginas web generadas con el generador de captchas. Estas páginas se almacenan en la ruta /ProyectoFinal_Compi1_2S24/Paginas_generadas pero para acceder a ellas desde la aplicación web es necesario servirlas. Para esto utilizo el comando **python3 -m http.server 8000** desde la carpeta donde se encuentran las páginas para iniciar un servidor web con el puerto 8000 y poder acceder a ellas a través de la dirección http://localhost:8000/captcha.html.

ANALIZADORES
CLANG Y CLC

CLANG

CLang es el lenguaje de etiquetado para la generación de captchas utilizando HTML y css inline. Permite la definición de elementos html con una lista de atributos opcionales para establecer propiedades y atributos del componente.

Tokens retornados

Símbolos	
Expresión regular	**Token retornado**
<	MENQUE
>	MAYQUE
[	CORPN
]	CORCL
=	IGUAL
“|”|comillas rectas	COMILL
/	BARRA

Etiquetas		
Expresión regular	**Token retornado**	**Uso**
C_CC	CC	Inicio de Archivo
C_HEAD	HEAD	Encabezado del documento
C_TITLE	TITLE	Título del documento
C_LINK	LINK	Enlace a otros archivos
C_BODY	BODY	Cuerpo del documento
C_SPAM	SPAM	Contenedor genérico
C_INPUT	INPUT	Controlador de entradas
C_TEXTAREA	TEXTAREA	Define un área de texto
C_SELECT	SELECT	Inicio de un componente con lista desplegable
C_OPTION	OPTION	Opción de la lista SELECT
C_DIV	DIV	Contenedor de más componentes

C_IMG	IMG	Imagen
C_BR	BR	Salto de línea
C_BUTTON	BUTTON	Botón común
C_H1	H1	Título común
C_P	P	Párrafo común
C_SCRIPTING	SCRIPTING	Indica un script

Parámetros		
Expresión regular	**Token retornado**	**Uso**
href	HREF	Referencia url externo
background	BACKGROUND	Color de fondo
color	COLOR	Color de texto
font-size	F_SIZE	Tamaño de texto en pixeles
font-family	F_FAM	Familia de fuente
text-align	TEXT_AL	Alinea el texto
type	TYPE	Tipo de entrada INPUT
id	ID	Identificador de componente
name	NAME	Nombre del componente
cols	COLS	Columnas de textArea
rows	ROWS	filas de textArea
class	CLASS	Como funcionara DIV
src	SRC	Url de una imagen
width	WIDTH	Ancho de imagen
height	HEIGHT	Alto de imagen
onclick()	ONCLICK	Acción que se ejecuta

Otros		
Expresión regular	**Token retornado**	**Uso**
"!!".*	-	Comentario de una línea
([\s\S]*?)	SCRIPT	Colecta toda la expresión script
"<!--"([\s\S]*?)"-->"	-	Comentario multilínea
[\r\t\n]	-	Espacios ignorados
[a-zA-Z0-9.:#/%_()]+	VALOR	Valor literal
<<EOF>>	EOF	Fin de archivo

Gramática

La gramática completa se encuentra desglosada en el archivo CLang.jison dentro de la carpeta analizadores del proyecto. Aquí solo se muestran producciones representativas para entender el funcionamiento de la gramática con etiquetas y parámetros modelo que describen de manera general el funcionamiento del resto de la gramática.

```
inicio ::= cc EOF

AUXILIARES - - - - - - - - - - - - - - - - - - - - - - - -
valores ::= VALOR valores | VALOR

PARÁMETROS - - - - - - - - - - - - - - - - - - - - - - - -
Se presentan modelos de los parámetros que sirven para entender el funcionamiento
de la gramática:

parametros ::= parametro parametros | /* nada */
parametro ::= href | font_size | type | cols | src | onclick | id
             | todos los demás parámetros mencionados en tabla parametros.
href ::= CORIZQ HREF IGUAL COMILL VALOR COMILL CORDER
        (href sirve de modelo para todos los demás parámetros)

Los valores retornados por los parámetros es el valor encerrado en comillas.

ETIQUETAS - - - - - - - - - - - - - - - - - - - - - - - - -
Se presentan modelos de etiquetas que sirven para entender el funcionamiento de la
gramática:

etiquetas ::= etiqueta etiquetas
             | etiqueta

etiqueta ::= head | title | div | spam | img | SCRIPT | cc | body | ... | error
             | todas las demás etiquetas descritas en la tabla de etiquetas
```

```
Algunas etiquetas contienen múltiples etiquetas dentro:
head ::= MENQUE HEAD MAYQUE etiquetas HEAD_FIN

Otras etiquetas no contienen más etiquetas dentro:
title ::= MENQUE TITLE MAYQUE valores TITLE_FIN

Algunas etiquetas contienen parámetros para definir propiedades o atributos
cc ::= MENQUE CC parámetros MAYQUE etiquetas CC_FIN

Los valores retornados por las etiquetas son objetos que representan al componente
al que la etiqueta hace alusión.
```

Funcionamiento de los parámetros

Un parámetro es una clase de javascript que contiene:

- **parámetro:** string en mayúsculas del tipo de parámetro que representa.
- **valor:** valor que tendrá el parámetro.

Las producciones de parámetros retornarán una instancia de la clase parámetro que será utilizada por las producciones que incluyan la producción **parámetros** en su definición.

El control de los parámetros se lleva a cabo en las instancias de las clases representantes de un componente HTML. Estas clases cuentan con una función para establecer parámetros según el tipo de componente que represente.

```
//Establecimiento de parámetros modelo (select) dentro de la producción jison
if(parametros != undefined && Array.isArray(parametros)){
    parametros.forEach(p=>{
        if(p!=undefined){
            try{
                select.establecerParametro(p.parametro, p.valor);
            }catch(error){
                 mostrarError(error);
            }
        }
    });
}
```

```
//Establecimiento de parámetros modelo dentro del componente (Link en este caso)
establecer Parametro(parametro, valor){
       switch(parametro){
           case 'HREF':
               if(this.href == undefined){
                   this.href = valor;
               }else{
                   var mensaje = "Error Semántico: se intento establecer el parámetro
                                  href pero ya estaba definido.";
                   super.lanzar Excepción Semántica(mensaje)
               }
               break;
           default:
               var mensaje = "Error Semantico: se intentó establecer el parametro "
                            + parametro + " en una instancia Link pero no es un
                             parametro valido para el componente.";
               super.lanzarExcepcionSemantica(mensaje);
       }
   }
```

Errores

Si se establece un parámetro en una etiqueta o producción que no contempla el uso de parámetros se retornará un error sintáctico que será controlada por la producción etiqueta que puede asumir el valor de un error.

Si se encuentra un error de sintaxis será controlado por la producción etiqueta, que puede asumir el valor de un error.

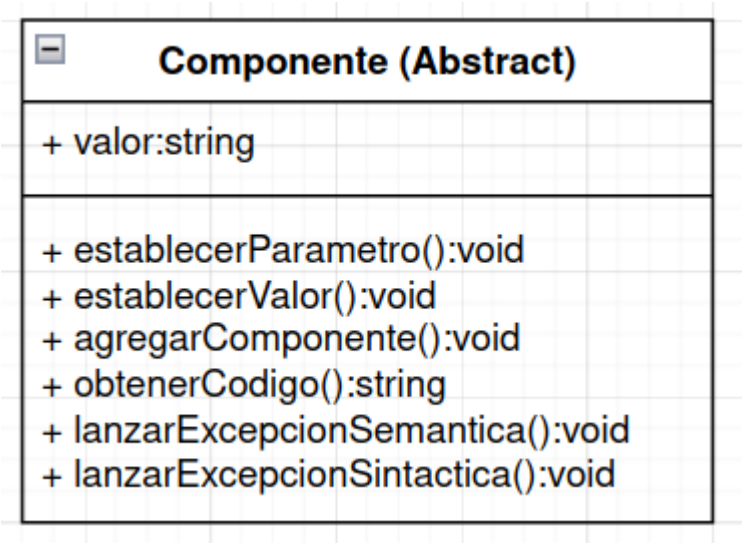
Si se establece un parámetro en una etiqueta que no contempla este parámetro se producirá un error semántico.

Si se intenta establecer un parámetro que ya fue previamente definido en una etiqueta se producirá un error semántico.

Clases representantes de componentes

Todos los componentes HTML aceptados por CLang tienen una clase para representar y manejar de mejor manera los valores y atributos de estos. Todos los componentes extienden de la clase Componente. A continuación se definen algunas clases representativas para entender su funcionamiento.

1. **Componente**



Esta clase representa un componente genérico para generar código HTML.

establecerParametro(parametro, valor):

- Permite establecer atributos HTML del componente
- Recibe el nombre del parámetro en mayúsculas y su valor
- Lanza un error si el parámetro no es válido o ya tiene un valor

establecerValor(valor):

- Define el contenido que irá entre las etiquetas HTML

agregarComponente(componente):

- Permite anidar componentes dentro de este componente permitiendo estructuras jerárquicas.

obtenerCodigo():

- Genera y retorna el código HTML final del componente

Métodos de manejo de errores:

- **lanzarExcepcionSemantica(mensaje):** Para errores de lógica/significado
- **lanzarExcepcionSintactica(mensaje):** Para errores de estructura/sintaxis
  Ambos imprimen el mensaje en consola y lanzan un Error

2. **Body**

Esta clase Body hereda de Componente y representa específicamente el elemento <body> de HTML. Analicemos su funcionalidad:

Propiedades:

- **background:** Almacena el color de fondo
- **componentes:** Array que guarda los componentes hijos

establecerParametro(parametro, valor):

- Solo acepta el parámetro 'BACKGROUND'
- Convierte nombres de colores en inglés a sus códigos hexadecimales correspondientes
- Permite valores hexadecimales directos si no coincide con ningún color predefinido

Lanza un error si:

- Se intenta establecer el background más de una vez
- Se intenta establecer cualquier otro parámetro que no sea 'BACKGROUND'

agregarComponente(componente):

- Agrega componentes hijos al array componentes
- Estos serán renderizados dentro del <body>

obtenerCodigo():

- Genera el código HTML del elemento body
- Establece background blanco (#FFFFFF) por defecto si no se especificó
- Concatena recursivamente el código de todos los componentes hijos

3. **Br**

Esta clase Br representa el elemento HTML
. Es una clase muy simple porque
 es un elemento vacío en HTML.

establecer Parametro(parametro, valor):

- Siempre lanza un error semántico indicando que no se pueden establecer parámetros en este componente

obtenerCodigo():

- Simplemente retorna "
"
- No necesita lógica adicional porque
 es un elemento vacío sin contenido ni cierre
- No requiere procesar componentes hijos ni atributos

4. **Button**

Esta clase Button representa un elemento <button> de HTML con estilos personalizables. Analicemos sus características:

Propiedades:

- color; // Color del texto
- fontSize; // Tamaño de fuente
- fontFamily; // Familia de fuente
- textAlign; // Alineación del texto
- id; // Identificador único
- background; // Color de fondo
- onclick; // Función para el evento click
- valor; // Texto del botón

establecerParametro(parametro, valor):

Acepta los siguientes parámetros:

- COLOR: Color del texto (acepta nombres predefinidos o códigos hex)
- FONT_SIZE: Tamaño de la fuente
- FONT_FAMILY: Tipo de fuente
- TEXT_ALIGN: Alineación (left, right, center, justify)
- ID: Identificador del botón
- BACKGROUND: Color de fondo
- ONCLICK: Función para el evento click

establecerValor(valor):

- Establece el texto que aparecerá en el botón

obtenerCodigo():

- Genera el código HTML del botón con todos sus estilos
- Establece valores por defecto si no se especificaron:
    - Color: negro (#000000)
    - Tamaño de fuente: 10px
    - Fuente: Arial
    - Alineación: izquierda
    - Fondo: blanco (#FFFFFF)
- Aplica estilos CSS predeterminados:
    - border-radius: 12px
    - padding: 15px 32px
- Retorna el botón HTML completo con todos sus atributos y estilos

5. **HTML**

Esta clase Html representa el elemento raíz de un documento HTML.

Propiedades:

- id // Identificador del documento
- name // Nombre del documento
- head // Componente Head
- body // Componente Body
- componentes = []// Array para otros componentes

establecerParametro(parametro, valor):

- Solo acepta 'ID' y 'NAME'
- Valida que no se establezcan más de una vez
- Lanza errores si los parámetros son inválidos o duplicados

Métodos para gestionar estructura:

- establecerHead(head): Configura la sección <head>
- establecerScript(script): Añade scripts al <head>
- establecerBody(body): Configura la sección <body>
- agregarComponente(componente): Añade componentes adicionales
- obtenerNombre(): Retorna el nombre del documento o "indefinido" si no está establecido

obtenerCodigo():

- Genera el documento HTML completo
- Incluye:
    - Declaración DOCTYPE
    - Elemento html con atributo lang="es"
    - Sección head (si está definida)
    - Sección body (si está definida)
    - Otros componentes agregados

6. **Select**

Extiende de Componente y representa un componente <select> de HTML que permite agregar opciones y establecer ciertos parámetros estilísticos.

Propiedades:

- **opciones**: Array que almacena los componentes de tipo Option que se agregarán al <select>.
- **fontSize, fontFamily, textAlign, color, id**: Parámetros que controlan el estilo y atributos del componente.

agregarComponente(componente):

- Agrega un objeto de tipo Option al array opciones, permitiendo crear una lista de opciones dentro del <select>.
-

establecerParametro(parametro, valor):

- Este método permite configurar diferentes parámetros estilísticos o de identificación (COLOR, FONT_SIZE, FONT_FAMILY, TEXT_ALIGN, ID) según el valor de parametro.
- Realiza validaciones para evitar que el mismo parámetro se establezca dos veces y lanza una excepción semántica si eso ocurre.
- Incluye una lista de colores comunes, mapeándolos a sus códigos hexadecimales.

obtenerCodigo():

- Genera el código HTML correspondiente al <select>, incluyendo las opciones en opciones y aplicando los estilos configurados.
- Si no se especifican los estilos, asigna valores por defecto como #000000 para el color y 10px para el tamaño de fuente.
- Recorre el array opciones y concatena el código HTML de cada opción, generando el código final de todo el componente <select>.

7. **Option**

Representa una opción (<option>) dentro de un elemento <select> y extiende de la clase Componente.

Propiedad valor:

- Almacena el texto que se mostrará dentro de la etiqueta <option> en HTML.

establecerParametro(parametro, valor):

- Este método es una implementación vacía en cuanto a la configuración de parámetros. Intenta asignar un parámetro, pero lanza una excepción indicando que Option no tiene parámetros configurables, mediante super.lanzarExcepcionSemantica.

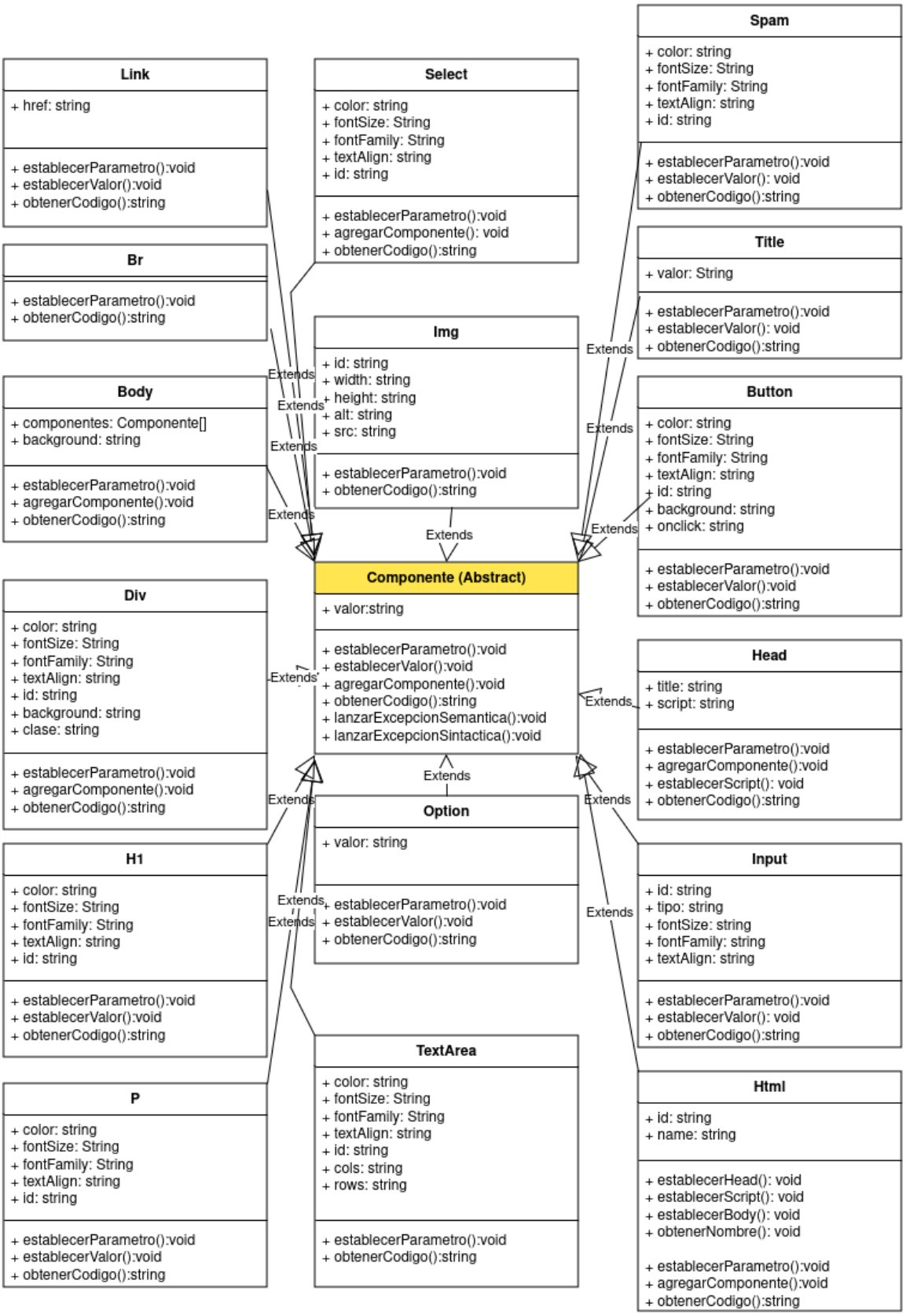
Método establecerValor(valor):

- Define el texto que aparecerá en la opción dentro del <select> al establecer this.valor.

Método obtenerCodigo():

- Devuelve el código HTML para la opción (<option>), utilizando this.valor para el texto, permitiendo que se use dentro de un elemento <select> generado dinámicamente.

Diagrama de clases



CLC

CLC es el lenguaje de scripting para la generación de un script en javascript que estará incrustado en el código html generado para el captcha. Cuenta con diversas producciones representando valores y expresiones además de estructuras de control.

Ignorar		
Expresión regular	**Token retornado**	**Uso**
[\r\t\n]	-	Espacios en blanco
<C_SCRIPTING>	-	Inicio script
</C_SCRIPTING>	-	Fin script
"!!".*	-	Comentario de una línea
"<!--"([\s\S]*?)"-->"	-	Comentario multilínea

Palabras reservadas		
Expresión regular	**Token retornado**	**Uso**
"FUNCTION_"?	FUNCTION	Declaración de función
“ON_LOAD”	ONLOAD	Declaración función onload
“@global”	GLOBAL	Modo de variable global
“IF”	IF	Inicio estructura if
“THEN”	THEN	-
“ELSE”	ELSE	Sentencia else
“REPEAT”	REPEAT	Inicio estructura repeat
“HUNTIL”	HUNTIL	Delimitador repeat
“WHILE”	WHILE	Inicio estructura while
“THENWHILE”	THENWHILE	Inicio bloque instrucciones while
“INIT”	INIT	Inicio bloque de instrucciones genérico
“END”	END	Fin bloque de instrucciones genérico
“INSERT”	INSERT	Función insert

Tipos de datos		
Expresión regular	**Token retornado**	**Uso**
“integer”	INT	Declaración entero
“string”	STR	Declaración string
“decimal”	DEC	Declaración decimal
“char”	CHR	Declaración char
“boolean”	BOO	Declaración boolean

Funciones especiales		
Expresión regular	**Token retornado**	**Uso**
“ASC”	ASC	Palabra ascendente
“DESC”	DESC	Palabra descendente
“LETPAR_NUM”	LETPAR_NUM	-
“LETIMPAR_NUM”	LETIMPAR_NUM	-
“REVERSE”	REVERSE	Palabra reversa
“CARACTER_ALEATORIO”	CARACTER_ALEATORIO	Función carácter aleatorio
“NUM_ALEATORIO”	NUM_ALEATORIO	Función número aleatorio
“getElementById”	GET_ELEMENT	Obtener elemento HTML
“ALERT_INFO”	ALERT_INFO	Mostrar alerta
“EXIT”	EXIT	Return
“REDIRECT”	REDIRECT	Redireccionar página
“INSERT”	INSERT	Insertar elementos html

Símbolos		
Expresión regular	**Token retornado**	**Uso**
(	PAROPN	-
)	PARCLS	-
[	COROPN	-
]	CORCLS	-
{	LLAVOPN	-
}	LLAVCLS	-
,	COMMA	-
;	SEMIC	-
:	COLON	-
=	EQU	-
+	MAS	-
-	MEN	-
/	DIV	-
*	TIM	-
!	EXC	-
<	MEN	-
>	MAY	-
|	ORS	-
&	AND	-

Valores		
Expresión regular	**Token retornado**	**Uso**
("\""([\s\S]*?)"\"")	STRING	Valor string primitivo
"""[a-zA-Z]"""\b	CHAR	Valor char primitivo
“true”	TRUE	Valor booleano primitivo
“false”	FALSE	Valor booleano primitivo
[0-9]+"."[0-9]+	DECIMAL	Valor decimal primitivo
[0-9]+\b	INTEGER	Valor entero primitivo
[a-zA-Z_$-][a-zA-Z0-9_$-]*	ID	Identificador

Otros		
Expresión regular	**Token retornado**	**Uso**
<<EOF>>	EOF	Fin de archivo

Gramática

La gramática completa se encuentra desglosada en el archivo CLC.jison dentro de la carpeta analizadores de la aplicación web. Aquí solo se muestran algunas producciones que sirven como modelo para entender el funcionamiento del parser a manera general.

```
inicio ::= functions EOF


valor ::= ID | INTEGER | primitivo | PAROPN condicion PARCLS
             | valor MAS valor | ...
             | valor TIM valor | ...
             | funcion_st
             | numero_aleatorio
funcion_st ::= asc | desc | letimpar_num | reverse | getElement

asc ::= ASC PAROPN STRING PARCLS
      | ASC PAROPN ID PARCLS

      Asc funciona como función modelo, algunas funciones string pueden recibir
      una cadena string directamente o una variable con un string
```

declaración modelo, funciona igual para los otros tipos de datos:

```
declaracion ::=  INT        identificadores
               | INT GLOBAL identificadores
               | INT        identificadores EQU ints
               | INT GLOBAL identificadores EQU ints
identificadores ::= ID identificadores
                  | ID
ints ::= numero COMMA ints
       | numero
numero ::= INTEGER
       | numero MAS numero
       | numero MEN numero
       | numero TIM numero
       | numero DIV numero
       | num_aleatorio
```

En caso de las declaraciones sin asignación, se recorre la lista de identificadores y se inicializan con su tipo y valor undefined. En caso de las declaraciones inicializadas, se recorren a la vez las listas de identificadores y de valores para declararse y asignarse de manera individual en orden. En cambio, si solo hay un valor disponible para asignar a multiples variables todas se inicializan con ese valor en específico.

```
condicion ::= valor AND AND valor
             | valor ORS ORS valor
             | EXCLAM valor
             | valor MAYQUE valor
             | ...
             | valor
```

la producción condición muestra modelos de condiciones, el parser retorna la condición construida en base a los valores y el operador.

```
asignacion ::= ID EQU valor
```

En caso de no haberse declarado el identificador en la tabla de símbolos se muestra el error.

Estructuras de control -
Pueden tener una sola expresión o contar con un bloque entero de instrucciones

```
if_exp::= IF PAROPN condicion PARCLS THEN expresion
        | IF PAROPN condicion PARCHL THEN bloque_ins
elif_exp::= ELSE if_exp
else_exp::= ELSE expresion
         |  ELSE bloque_ins
```

```
while_exp::= WHILE PAROPN condicion PARCLS THENWHILE expresion
           | WHILE PAROPN condicion PARCLS THENWHILE bloque_ins
```

Bloques de expresiones / instrucciones - - - - - - - -
Pueden ser un conjunto de instrucciones una tras otra o estar delimitada dentro de un bloque.

```
expresiones ::= expresion expresiones
              | expresion
expresion ::= declaracion SEMIC
              | asignacion SEMIC
              | if_exp
              | repeat
              | while_exp
              | alert_info
              | exit
              | redirect
              | insert
              | error
bloque_ins ::= INIT LLAVOP COLON expresiones COLON LLAVCL END
```

El manejo de errores puede realizarse en la producción de funciones o de expresión.

```
function ::= FUNCTION ID PAROPN PARCLS expresiones CORCLS
            | ONLOAD     PAROPN PARCLS expresiones CORCLS
            | error
```

Funciones especiales -
Estas funciones realizan una acción preestablecida, varían en cuanto a su declaración

```
alert_info: ALERT_INFO PAROPN STRING PARCLS SEMIC
       | ALERT_INFO PAROPN ID     PARCLS SEMIC
exit : EXIT PAROPN PARCLS SEMIC
redirect : REDIRECT PAROPN PARCLS SEMIC
insert : INSERT PAROPN inserts PARCLS
inserts : STRING | STRING COMMA inserts | STRING inserts
       | ID   | ID COMMA inserts | ID insert
```

Tabla de símbolos

La tabla de símbolos es manejada por el parser para llevar un control de las variables declaradas así como los valores que van tomando a lo largo de la ejecución.

La declaración de los símbolos puede ser inicializada o no inicializada.

- Declaración no inicializada: inicia una variable aumentando la posición del símbolo, el identificador que tendrá la variable, tipo, valor indefinido, modo, ambito y se especifica que el valor (indefinido en este caso) es por declaración
- Declaración inicializada: Es igual a la declaración no inicializada, con la diferencia de que se establece el valor correspondiente indicando que fue asignado por declaración.

Además, la tabla de símbolos lleva un control de cambio de valor de la variable, indicando que el cambio se realizó por asignación en este caso.

Un ejemplo gráfico de la tabla de símbolos es el siguiente:

Tabla de simbolos

Pos.	Identificador	Tipo	Valor actual	Modo	Ambito	Por
1	contador_fallas	integer	5	@global	FUNCTION_calc()	declaracion
2	result_caja_texto	string	getElementById("entrada_1")	-	FUNCTION_calc()	declaracion
3	result	string	"10 "	@global	FUNCTION_calc()	declaracion
4	result		"30 "		FUNCTION_calc()	asignacion
5	no_declarada		40		FUNCTION_calc()	NO DECLARADO
6	p_declaracion	string	undefined	@global	FUNCTION_calc()	declaracion
7	p2_declaracion	string	undefined	-	FUNCTION_calc()	declaracion
8	mensaje_fallo	string	"El captcha no fue validado intente otra vez "	-	FUNCTION_calc()	declaracion
9	mensaje_acierto	string	"El captcha fue validado "	-	FUNCTION_calc()	declaracion
10	mensaje_final	string	"El captcha no logró ser validado :(intente mas tarde"	-	FUNCTION_calc()	declaracion

Aquí se puede observar las declaraciones, el control de valores, ámbitos de ejecución y errores en caso de que se intenten asignar valores a variables que no han sido previamente inicializadas.

Generación de código javascript

Cada instrucción reconstruye el código javascript equivalente en su producción, tomando en cuenta que no existan errores en la producción (en caso de existir, se ignora la instrucción en conflicto).

Las instrucciones reconstruidas son elevadas a un nivel superior para su adición en las producciones que están un nivel más arriba hasta llegar a la producción de la función completa.

Todo el código es almacenado en una variable codigoScript en el analizador, mismo que será tomado posteriormente por el codigo TypeScript del componente para ser implementado en la clase HTML y así completar el archivo para ser almacenado gracias al servidor.