

**Github:** <https://github.com/kelok3rik/Tarea-Semana-7>

## PRUEBA

The screenshot displays a Java compiler interface with four main panels: **Texto del Archivo**, **Tabla de Símbolos**, **Código Intermedio**, and **Código Java**.

**Texto del Archivo:** Contains a BASIC-like code snippet for a program named 'Facturar'. It includes variable declarations (NUM, DNUM, WORD), input/output statements (LEER, ESCRIBIR), and arithmetic/logic operations (CUANDO, SI, LEER, LEER).

**Tabla de Símbolos:** A table listing tokens and their counts. The tokens are categorized into Palabra Reservada (Reserved Word), Palabra (Word), and Delimitador (Delimiter).

| Token    | Cantidad | Tipo        |
|----------|----------|-------------|
| WORD     | 1        | Palabra ... |
| DNUM     | 1        | Palabra ... |
| LEER     | 4        | Palabra ... |
| FCUANDO  | 1        | Palabra ... |
| NUM      | 1        | Palabra ... |
| FINAL    | 1        | Palabra ... |
| CUANDO   | 1        | Palabra ... |
| ESCRIBIR | 6        | Palabra ... |
| SI       | 2        | Palabra ... |
| INICIO   | 1        | Palabra ... |
| FSI      | 2        | Palabra ... |
| *        | 1        | Operador    |
| +        | 12       | Operador    |
| <        | 2        | Operador    |
| -        | 1        | Operador    |
| =        | 10       | Operador    |
| >        | 1        | Operador    |
| /        | 1        | Operador    |
| #        | 20       | Delimita... |
| {        | 13       | Delimita... |
| }        | 13       | Delimita... |

**Código Intermedio:** Shows the intermediate code generated from the input, including variable declarations, input/output statements, and arithmetic/logic operations.

**Código Java:** Shows the final Java code generated from the intermediate code, including imports, class declarations, and the main logic of the program.

**Listado de errores:** A section for displaying any compilation errors.

**Botones de control:** Tabla de símbolos, Análisis Sintáctico - Semántico, Traducir, and Limpiar.

## PALABRAS RESERVADAS

```
r.put(key: "INICIO", value: 0);
r.put(key: "FINAL", value: 0);
r.put(key: "WORD", value: 0);
r.put(key: "ALFA", value: 0);
r.put(key: "NUM", value: 0);
r.put(key: "DNUM", value: 0);
r.put(key: "BOOL", value: 0);
r.put(key: "LNUM", value: 0);
r.put(key: "LEER", value: 0);
r.put(key: "ESCRIBIR", value: 0);
r.put(key: "CUANDO", value: 0);
r.put(key: "SI", value: 0);
r.put(key: "IS", value: 0);
r.put(key: "DESDE", value: 0);
r.put(key: "PASO", value: 0);
r.put(key: "HASTA", value: 0);
r.put(key: "FDESDE", value: 0);
r.put(key: "FCUANDO", value: 0);
r.put(key: "FSI", value: 0);

op.put(key: "/", value: 0);
op.put(key: "*", value: 0);
op.put(key: "+", value: 0);
op.put(key: "-", value: 0);
op.put(key: "=", value: 0);
op.put(key: "^", value: 0);
op.put(key: "<", value: 0);
op.put(key: ">", value: 0);
op.put(key: "||", value: 0);
op.put(key: "&&", value: 0);

deli.put(key: "#", value: 0);
deli.put(key: ";", value: 0);
deli.put(key: "{", value: 0);
deli.put(key: "}", value: 0);
deli.put(key: "(", value: 0);
deli.put(key: ")", value: 0);
```

## DOCUMENTACION

### Boton “Tabla de símbolos”

- **Inicialización de Estructuras de Datos:**  
Se crean varias estructuras de datos (HashMap y LinkedList) para almacenar la información sobre los tokens encontrados. Cada HashMap está destinado a un tipo específico de token, y la LinkedList llamada texto se utiliza para almacenar textos entre delimitadores '#’.
- **Conteo de Ocurrencias:**  
Se utiliza un bucle while junto con la clase StringTokenizer para recorrer cada token en el texto de entrada. Se cuentan las ocurrencias de palabras reservadas, operadores, delimitadores, identificadores, números y textos.
- **Manejo de Delimitadores y Textos:**  
Cuando se encuentra el delimitador #, se espera que el siguiente token sea un texto. Se recopila el texto hasta que se encuentra otro #. El texto se agrega a la lista texto, y su ocurrencia se cuenta.
- **Construcción del Modelo para la Tabla:**  
Se utiliza un modelo de tabla (DefaultTableModel) para organizar la información recopilada. Se añaden filas al modelo para cada tipo de token encontrado, especificando el token, la cantidad de ocurrencias y el tipo de token.

### Boton “Buscar errores”

- **Expresiones Regulares:**  
Se definen varias expresiones regulares para patrones que se utilizan en la verificación del código. Por ejemplo, se definen patrones para identificadores (id), números enteros (entero), números decimales (decimal), condiciones (condicion), y otros.
- **Listas y Variables:**  
Se crean varias listas (ENT, DEC, TEXT, TAKE) para almacenar identificadores de diferentes tipos (numéricos, decimales, de texto, de entrada) a medida que se encuentran durante el análisis.
- **Tokens y Contadores:**  
Se utiliza un bucle while junto con la clase StringTokenizer para recorrer cada token en el texto de entrada. Se cuentan los errores y se llevan a cabo comprobaciones específicas según el tipo de token encontrado.
- **Manejo de Estructuras de Control:**  
Se realizan verificaciones para las estructuras de control del lenguaje, como ciclos (DESDE y FDESDE), condiciones (CUANDO y FCUANDO), y estructuras condicionales (SI y FSI).
- **Declaración de Variables:**  
Se verifica la declaración y uso correcto de variables, incluida la comprobación de errores semánticos, como el uso de variables no declaradas.

- **Errores y Mensajes:**

Se utilizan variables (errores, Error, LineaError) para realizar un seguimiento de los errores encontrados durante el análisis. Además, se actualiza la interfaz de usuario con mensajes de error.