Informe Técnico - JavaLang Interpreter

Índice

Gramática Formal de Javalang

```
%start programa;

programa: lista_declaraciones;

lista_declaraciones: lista_declaraciones declaracion
                   | declaracion
                   ;

// Reglas para tipos primitivos
declaracion:    tipoPrimitivo corchetes_lista TOKEN_IDENTIFIER '('
lista_parametros_opt ')' bloque
                | declaracion_funcion
                | declaracion_main
```

```
                | tipoPrimitivo corchetes_lista TOKEN_IDENTIFIER var_tail ';'
                | TOKEN_FINAL tipoPrimitivo corchetes_lista TOKEN_IDENTIFIER var_tail
';'
                ;

declaracion_funcion:    TOKEN_VOID TOKEN_IDENTIFIER '(' lista_parametros_opt ')'
bloque
                        ;

lista_parametros_opt:   lista_parametros
                        | %empty
                        ;

lista_parametros:   lista_parametros ',' parametro
                    | parametro
                    ;

parametro:  tipoPrimitivo corchetes_lista TOKEN_IDENTIFIER corchetes_lista
            ;

declaracion_main:   TOKEN_PUBLIC TOKEN_STATIC TOKEN_VOID TOKEN_MAIN
                    '(' TOKEN_DSTRING '[' ']' TOKEN_IDENTIFIER ')' bloque
                    ;

lista_argumentos_opt:   lista_Expr
                        | %empty
                        ;

lSentencia: lSentencia sentencia
            | sentencia
            ;

sentencia:  bloque
            | declaracion_var ';'
            | if_stmt
            | while_stmt
            | for_stmt
            | switch_stmt
            | TOKEN_BREAK ';'
            | TOKEN_CONTINUE ';'
            | TOKEN_RETURN expr ';'
            | TOKEN_RETURN ';'
            | expr ';'
            | error ';'
            ;

lista_Expr: lista_Expr ',' lista_item
            | lista_item

lista_item: expr
            | inicializador_arreglo
            ;

bloque: '{' lSentencia '}'
        | '{' '}'
```

```
        ;

declaracion_var:    tipoPrimitivo declarador_completo
                    | TOKEN_FINAL tipoPrimitivo declarador_completo
                    ;


declarador_completo: corchetes_lista TOKEN_IDENTIFIER corchetes_lista '='
inicializador_arreglo
                    | corchetes_lista TOKEN_IDENTIFIER corchetes_lista '=' expr
                    | corchetes_lista TOKEN_IDENTIFIER corchetes_lista
                    ;

corchetes_lista:    corchetes_lista '[' ']'
                    | %empty
                    ;

inicializador_arreglo : '{' lista_Expr '}'
                        | '{' '}'
                        ;

// Cola de declarador para nivel superior (dimensiones derechas y opcional
inicializador)
var_tail: corchetes_lista '=' inicializador_arreglo
        | corchetes_lista '=' expr
        | corchetes_lista
        ;


asignacion_expr: TOKEN_IDENTIFIER '=' expr
                | primary_expr '[' expr ']' '=' expr
                | TOKEN_IDENTIFIER TOKEN_PLUS_ASSIGN expr
                | TOKEN_IDENTIFIER TOKEN_MINUS_ASSIGN expr
                | TOKEN_IDENTIFIER TOKEN_MULT_ASSIGN expr
                | TOKEN_IDENTIFIER TOKEN_DIV_ASSIGN expr
                | TOKEN_IDENTIFIER TOKEN_MOD_ASSIGN expr
                | TOKEN_IDENTIFIER TOKEN_AND_ASSIGN expr
                | TOKEN_IDENTIFIER TOKEN_OR_ASSIGN expr
                | TOKEN_IDENTIFIER TOKEN_XOR_ASSIGN expr
                | TOKEN_IDENTIFIER TOKEN_LSHIFT_ASSIGN expr
                | TOKEN_IDENTIFIER TOKEN_RSHIFT_ASSIGN expr
                | TOKEN_IDENTIFIER TOKEN_URSHIFT_ASSIGN expr
                ;


expr: asignacion_expr
    | expr TOKEN_OR expr
    | expr TOKEN_AND expr
    | expr '|' expr
    | expr '^' expr
    | expr '&' expr
    | expr TOKEN_IGUAL_IGUAL expr
    | expr TOKEN_DIFERENTE expr
    | expr '<' expr
    | expr '>' expr
```

```
    | expr TOKEN_MENOR_IGUAL expr
    | expr TOKEN_MAYOR_IGUAL expr
    | expr TOKEN_LSHIFT expr
    | expr TOKEN_RSHIFT expr
    | expr TOKEN_URSHIFT expr
    | expr '+' expr
    | expr '-' expr
    | expr '*' expr
    | expr '/' expr
    | expr '%' expr
    | primary_expr
    ;


primary_expr: '(' expr ')'
            | '(' tipoPrimitivo ')' primary_expr
            | TOKEN_IDENTIFIER
            | primary_expr '[' expr ']'
            | expresion_creacion_arreglo
            | TOKEN_PARSE_INT '(' expr ')'
            | TOKEN_PARSE_FLOAT '(' expr ')'
            | TOKEN_PARSE_DOUBLE '(' expr ')'
            | TOKEN_STRING_VALUEOF '(' expr ')'
            | TOKEN_STRING_JOIN '(' expr ',' lista_Expr ')'
            | TOKEN_ARRAYS_INDEXOF '(' expr ',' expr ')'
            | primary_expr TOKEN_DOT TOKEN_LENGTH
            | TOKEN_ARRAY_ADD '(' expr ',' expr ')'
            | TOKEN_IDENTIFIER '(' lista_argumentos_opt ')'
            | TOKEN_PRINT '(' lista_Expr ')'
            | primitivo
            | '-' primary_expr %prec NEG
            | '!' primary_expr
            | '~' primary_expr
            | primary_expr TOKEN_INCREMENTO
            | primary_expr TOKEN_DECREMENTO
            | primary_expr TOKEN_DOT TOKEN_IDENTIFIER '(' expr ')'
            ;

primitivo: TOKEN_INTEGER
    | TOKEN_HEX_INTEGER
    | TOKEN_FLOAT_LIT
    | TOKEN_DOUBLE_LIT
    | TOKEN_STRING_LITERAL
    | TOKEN_CHAR_LITERAL
    | TOKEN_TRUE
    | TOKEN_FALSE
    ;


tipoPrimitivo: TOKEN_DINT
    | TOKEN_DFLOAT
    | TOKEN_DDOUBLE
    | TOKEN_DSTRING
    | TOKEN_DCHAR
    | TOKEN_DBOOLEAN
```

```
    ;


if_stmt: TOKEN_IF '(' expr ')' bloque %prec IFX
        | TOKEN_IF '(' expr ')' bloque TOKEN_ELSE bloque
        | TOKEN_IF '(' expr ')' bloque TOKEN_ELSE if_stmt {
        ;


while_stmt: TOKEN_WHILE '(' expr ')' bloque;


for_init: declaracion_var
    | expr            ;

for_expr_opt: expr
            | %empty
            ;


for_stmt:
    TOKEN_FOR '(' for_head ')' bloque
    ;


for_head:
    tipoPrimitivo corchetes_lista TOKEN_IDENTIFIER corchetes_lista ':' expr
    | tipoPrimitivo corchetes_lista TOKEN_IDENTIFIER corchetes_lista '='
inicializador_arreglo ';' for_expr_opt ';' for_expr_opt
    | tipoPrimitivo corchetes_lista TOKEN_IDENTIFIER corchetes_lista '=' expr ';'
for_expr_opt ';' for_expr_opt
    | tipoPrimitivo corchetes_lista TOKEN_IDENTIFIER corchetes_lista ';' for_expr_opt
';' for_expr_opt
    | for_init_opt ';' for_expr_opt ';' for_expr_opt
    ;


for_init_opt: for_init
    | %empty
    ;


switch_stmt: TOKEN_SWITCH '(' expr ')' '{' case_list '}' ;



case_list: case_list case_stmt
        | case_stmt
        ;

case_stmt:  TOKEN_CASE expr ':' lSentencia_opt
            | TOKEN_DEFAULT ':' lSentencia_opt
            ;
```

```
lSentencia_opt: lSentencia
                | %empty
                ;


expresion_creacion_arreglo: TOKEN_NEW tipoPrimitivo lista_dims_expr dims_vacias
                          | TOKEN_NEW tipoPrimitivo lista_dims_expr
                          ;

lista_dims_expr: lista_dims_expr '[' expr ']'
                | '[' expr ']'
                ;

dims_vacias
    : dims_vacias '[' ']'
    | '[' ']'
    ;
```

Diseño de Módulos

La arquitectura del intérprete sigue el flujo de un compilador:

Modulo de Análisis Léxico (Flex)

Este módulo tiene la responsabilidad de leer el archivo de código fuente (.java) como texto plano y descomponerlo en una secuencia de unidades lógicas mínimas, llamadas tokens. En este módulo se definieron patrones mediante expresiones regulares que identifican tokens del lenguaje y cada patrón se asoció con un token numérico que se entrega al analizador sintáctico.

Módulo de Análisis Sintáctico (Bison)

Este módulo tiene la responsabilidad de recibir la secuencia de tokens del analizador léxico y verificar si sigue las reglas gramaticales del lenguaje JavaLang. Su objetivo principal es construir un Árbol de Sintaxis Abstracta (AST) que represente la estructura jerárquica del código. En este modulo se crearon las reglas para la contruscción del lenguaje y a medida de que se va realizando este análisis, se crean nodos correspondientes al AST y se enlazan para crear el arbol completo.

Módulo del Árbol de Sintaxis Abstracta (AST)

Este módulo tiene la responsabilidad servir como la estructura de datos central que representa el código fuente de una manera lógica y jerárquica, facilitando su posterior interpretación. En este módulo se diseñó un `struct` base que contiene campos comunes para todos los nodos y un puntero `Result (*interpret)(AbstractExpresion*, Context*)`. Con este patrón de diseño cada nodo tiene su propia lógica de interpretación.

Módulo de Contexto y Tabla de Símbolos

Este módulo tiene la responsabilidad gestionar los ámbitos del programa y almacenar información sobre las variables y funciones declaradas. En este módulo se crea la estructura `struct Context` que representa un ámbito y que contiene el puntero a su contexto padre. Con esto se forma una lista enlazada que simula la pila de llamdas. Luego dentro de cada lista hay `struct Symbol` que se encarga de almacenar el nombre, tipo, valor y si es una contante.

Desafíos Enfrentados

Durante el desarrollo del intérprete presetó varios desafios complejos que requirieron un análisis profundo y soluciones precisas.

Ambigüedad Gramatical: `for` vs. `forEach`

Problema

Al introducir el bucle `forEach`, la gramática se volvió ambigua. Cuando el parser leía `for (int i...`, no podía decidir si se trataba de un `for` clásico `(for (int i = 0;...)` o un `forEach (for (int i :...)` basándose en un solo token de anticipación.

Solución

Se creó una regla no terminal intermedia `(for_head)` que se encarga exclusivamente de analizar el contenido dentro de los paréntesis del `for`. Esta regla fuerza a Bison a buscar el token decisivo antes de comprometerse con una de las dos producciones.

Gestión de Memoria: El Bug de la "Fotocopia Fantasma"

Problema

Las modificaciones a los elementos de un arreglo dentro de un bucle `for` o en un arreglo final no se veían reflejadas. Por ejemplo, `DIAS[1] = "Martes-Modificado";` no cambiaba el valor.

Solución

Se rediseñó por completo la lógica de asignación de arreglos. En lugar de simplemente interpretar el lado izquierdo de la asignación, la nueva función navega por el AST para encontrar el nombre de la variable base, la busca directamente en la tabla de símbolos para obtener un puntero al `ArrayValue` original, y realiza la modificación sobre la estructura de datos real.

Propagación de Señales de Control (break, continue)

Problema

Después de ejecutar un `forEach` que contenía una sentencia `continue`, el resto del programa dejaba de ejecutarse.

Solución

Se refinó la lógica de retorno de todas las estructuras de bucle. Ahora, las señales como `continue` y `break` son consumidas y limpiadas dentro del bucle que las genera. Solo las señales que deben propagarse a un nivel superior se devuelven.

Resultados de Pruebas y Métricas de Cobertura

Para validar el correcto funcionamiento de todo, se realizaron distintos archivo de prueba dentro de la carpeta `test` para corroborar de que todo estuviera funcionando bien. Entre estos archivos de entrada se encuentra uno de los más importantes que seria el que se llama `rubrica.usl`:

```
// Función para probar Recursividad
int factorial(int n) {
    if (n <= 1) {
        return 1;
    }
    return n * factorial(n - 1);
}

// Función void para probar llamadas sin retorno
void imprimirSeparador() {
    System.out.println("----------------------------------------");
}

// Función que prueba todas las estructuras de control
void probarEstructurasDeControl() {
    System.out.println("\n--- 2. Pruebas de Estructuras de Control ---");

    // Prueba de IF-ELSE IF-ELSE
    int calificacion = 85;
    if (calificacion > 90) {
        System.out.println("Resultado: Excelente");
    } else if (calificacion > 70) {
        System.out.println("Resultado: Bueno"); // <-- Debería imprimir esto
    } else {
        System.out.println("Resultado: Necesita mejorar");
    }

    // Prueba de SWITCH con CASE, BREAK y DEFAULT
    char opcion = 'B';
    switch (opcion) {
        case 'A': System.out.println("Switch: Opcion A"); break;
        case 'B': System.out.println("Switch: Opcion B"); break; // <-- Debería
imprimir esto
        default: System.out.println("Switch: Opcion Default"); break;
    }

    // Prueba de WHILE con CONTINUE
    int i = 0;
    String resultadoWhile = "";
    while (i < 5) {
        i = i + 1;
        if (i == 3) {
            continue; // Saltar el número 3
        }
        resultadoWhile = resultadoWhile + i;
    }
    System.out.println("While (sin el 3): " + resultadoWhile); // Esperado: 1245

    // Prueba de FOR CLÁSICO con BREAK
    int ultimoI = 0;
    for (int j = 0; j < 10; j = j + 1) {
        if (j == 5) {
            break; // Detener en 5
        }
        ultimoI = j;
```

```
    }
    System.out.println("For clasico se detuvo en: " + ultimoI); // Esperado: 4
}


public static void main(String[] args) {
    System.out.println("====== INICIO: Rúbrica de Calificación del Intérprete
======");

    // --- 1. VARIABLES, TIPOS, CONSTANTES Y OPERADORES ---
    imprimirSeparador();
    System.out.println("--- 1. Variables, Tipos, Constantes y Operadores ---");
    final int CONSTANTE_FINAL = 100;
    int a = 20;
    double b = 5.5;
    boolean c = true;
    String d = "Hola";
    char e = 'A';

    // Aritméticos y Relacionales
    double calculo = (a + CONSTANTE_FINAL) / 10.0 * (b - 0.5); // (120 / 10.0) * 5.0
= 12.0 * 5.0 = 60.0
    System.out.println("Calculo aritmetico: " + calculo);
    System.out.println("Comparacion: " + (calculo == 60.0));

    // Lógicos y Bitwise
    int flags = 12; // 1100
    int mascara = 10; // 1010
    if ((flags & mascara) == 8 && (flags | mascara) == 14) { // AND=1000 (8), OR=1110
(14)
        System.out.println("Operadores Bitwise OK.");
    }

    // --- 2. ESTRUCTURAS DE CONTROL ---
    imprimirSeparador();
    probarEstructurasDeControl();

    // --- 3. ARREGLOS (MULTIDIMENSIONALES Y PROPIEDADES) ---
    imprimirSeparador();
    System.out.println("\n--- 3. Arreglos (Multidimensionales y Propiedades) ---");
    final String[][] agenda = {{"Juan", "555-1234"}, {"Ana", "555-5678"}};
    agenda[1][1] = "555-8765"; // Modificar contenido de arreglo final (DEBE
funcionar)
    System.out.println("Nuevo telefono de Ana: " + agenda[1][1]);

    // Prueba de .length en arreglo multidimensional
    System.out.println("Numero de contactos: " + agenda.length);

    // Prueba de forEach anidado
    String contactos = "";
    for (String[] contacto : agenda) {
        contactos = contactos + contacto[0] + "; ";
    }
    System.out.println("Nombres: " + contactos);
```

```
    // --- 4. FUNCIONES (LLAMADAS Y RECURSIVIDAD) ---
    imprimirSeparador();
    System.out.println("\n--- 4. Funciones (Llamadas y Recursividad) ---");
    System.out.println("Llamada a factorial(5): " + factorial(5));

    // --- 5. FUNCIONES EMBEBIDAS ---
    imprimirSeparador();
    System.out.println("\n--- 5. Funciones Embebidas ---");
    String[] partes = {"Prueba", "de", "Join"};
    String joined = String.join("-", partes);
    System.out.println("String.join: " + joined);

    int[] numeros = {10, 20, 30, 20, 40};
    System.out.println("Arrays.indexOf(20): " + Arrays.indexOf(numeros, 20));

    numeros = Array.add(numeros, 50);
    System.out.println("Array.add: Nuevo tamaño: " + numeros.length);

    // --- 6. CASTEO Y STRINGS ---
    imprimirSeparador();
    System.out.println("\n--- 6. Casteo y Strings ---");
    double valorDoble = 99.9;
    int valorEntero = (int)valorDoble; // Casteo explícito
    System.out.println("Casteo de 99.9 a int: " + valorEntero);

    String str1 = "Java";
    String str2 = "Lang";
    String str3 = str1 + str2;
    System.out.println("Concatenacion: " + str3);
    System.out.println("Comparacion .equals(): " + str3.equals("JavaLang"));

    System.out.println("\n====== FIN: Rúbrica de Calificación del Intérprete
======");
}
```

Al correrlo correctamente se dennotó que el programa funcionaba correctamente.