Manual Técnico

PROYECTO 2

Juan Marcos Ibarra López | OLC1 | 22/05/2020
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

INTRODUCCIÓN

La aplicación fue realizada con el propósito de encontrar copias entre archivos en el lenguaje JAVA.

CONTENIDO TECNICO

ANALIZADOR LEXICO

El analizador léxico se implementó en la aplicación, para que se pueda mostrar correctamente las palabras reservadas, además, si el usuario ingresa un texto con algún error el analizador léxico le ayudará a encontrar el error cometido de una manera mucho más sencilla y eficaz. Toda la implementacion se manejó con la herramienta JISON.

LOGICA UTILIZADA

```
%options case-sensitive
%%
\s+                                     // se ignoran espacios en blanco
"//".*                                  // comentario simple línea
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]     // comentario multiple líneas

// PALABRAS RESERVADAS

//TIPOS DE DATOS
"int"                           return 'INT';
"char"                          return 'CHAR';
"String"                        return 'STRING';
"double"                        return 'DOUBLE';
"boolean"                       return 'BOOLEAN';

";"                             return 'PUNTO_COMA';
","                             return 'COMA';
":"                             return 'DOS_PUNTOS';

"++"                            return 'INCREMENTO';
"--"                            return 'DECREMENTO';
//OPERADORES

">="                            return 'MAYOR_IGUAL';
"<="                            return 'MENOR_IGUAL';
">"                             return 'MAYOR';
"<"                             return 'MENOR';
"!="                            return 'DIFERENTEA';
"=="                            return 'DOBLE_IGUAL';
"!"                             return 'NOT';
"="                             return 'IGUAL';
"||"                            return 'OR';
"&&"                            return 'AND';
// OPERADORES
```

```
// OPERADORES
"+"                                   return 'SUMA';
"-"                                   return 'RESTA';
"/"                                   return 'DIVISION';
"*"                                   return 'MULTIPLICACION';
"^"                                   return 'POTENCIA';
"%"                                   return 'MODULO';

"("                                   return 'PARENTESIS_APERTURA';
")"                                   return 'PARENTESIS_CIERRE';
"{"                                   return 'LLAVE_APERTURA';
"}"                                   return 'LLAVE_CIERRE';
//RESERVADAS DE JAVA
"true"                                return 'TRUE';
"false"                               return 'FALSE';
"class"                               return 'CLASS';
"import"                              return 'IMPORT';
"continue"                            return 'CONTINUE';
"void"                                return 'VOID';
"return"                              return 'RETURN';
"main"                                return 'MAIN';

"if"                                  return 'IF';
"else"                                return 'ELSE';
"while"                               return 'WHILE';
"do"                                  return 'DO';
"for"                                 return 'FOR';
"switch"                              return 'SWITCH';
"case"                                return 'CASE';
"default"                             return 'DEFAULT';
"break"                               return 'BREAK';
"System.out.println"                  return "IMPRIMIR";
"System.out.print"                    return "IMPRIMIR";
```

```
\"([^\\\"]|\\.)*\"      { yytext = yytext.substr(1,yyleng-2); return 'CADENA'; }
\'[^\']*\'              { yytext = yytext.substr(1,yyleng-2); return 'CARACTER'; }
[0-9]+("."[0-9]+)?\b    return 'DECIMAL';
[0-9]+\b                return 'ENTERO';
([a-zA-Z_])[a-zA-Z0-9_]* return 'IDENTIFICADOR';

<<EOF>>                 return 'EOF';
.                       { console.error('Este es un error léxico: ' + yytext + ', en la linea: ' + yylloc.first_line + ', en la columna: ' + yylloc.first_column);
                          instruccionesAPI.pushError(instruccionesAPI.errorLexico(yytext,yylloc.first_line,yylloc.first_column));     }
```

ANALIZADOR SINTACTICO

Para realizar el analizador sintáctico primero se procedió a crear las gramáticas para poder hacer un analizador sintáctico descendente.

ALGUNAS GRAMATICAS IMPLEMENTADAS

```
ini
    : Imports ClassINIT  EOF
        {  return {
            AST: instruccionesAPI.instructionsINIT($1,$2),
            ListaErrores: instruccionesAPI.getListaErrores()};}
    | Error ClassINIT EOF
        { return {
            AST: instruccionesAPI.instructionsINIT(undefined,$1),
            ListaErrores: instruccionesAPI.getListaErrores()
        };}
    | ClassINIT EOF
        { return {
            AST: instruccionesAPI.instructionsINIT(undefined,$1),
            ListaErrores: instruccionesAPI.getListaErrores()
        };}
;

ClassINIT
    : Class { $$ = [$1] }
    | ClassINIT Class  {  $1.push($2); $$ = $1 }
        //  instruccionesAPI.pushError(instruccionesAPI.errorSintactico(yytext,yy.parser.hash.expected,this._$.first_line, this._$.f
;
Class
    : CLASS IDENTIFICADOR LLAVE_APERTURA InstruccionesDentroClase LLAVE_CIERRE { $$ = instruccionesAPI.instructionClass($2,$4) }
    | CLASS IDENTIFICADOR LLAVE_APERTURA LLAVE_CIERRE { $$ = instruccionesAPI.instructionClass($2,undefined) }
;
InstruccionesDentroClase
    : InstruccionesDentroClase Instruccion_InsideClass {  $1.push($2); $$ = $1 }
    | Instruccion_InsideClass { $$ = [$1] }
;
```

```
Instruccion_InsideClass
    : Declaracion
    | FuncionMetodo
    | Clase
    | Error
        // instruccionesAPI.pushError(instruccionesAPI.errorSintactico(yytext,yy.parser
//    | error TokEnd{ console.error('Este es un error sintáctico: ' + yytext + ', en la
;

Instruccion_Functions
    : Declaracion
    | Asignacion
    | If
    | For
    | While
    | Do
    | Switch
    | Imprimir
    | Class
    | BREAK PUNTO_COMA { $$ = instruccionesAPI.instructionBreak() }
    | CONTINUE PUNTO_COMA { $$ = instruccionesAPI.instructionContinue() }
    | Return
    | LlamarFuncion PUNTO_COMA
    | Error

;
Declaracion
    :Tipo_Dato Declaracion1 PUNTO_COMA { $$ = instruccionesAPI.declaration0($1,$2) }
;
Declaracion1
```

```
Tipo_Dato
    :INT
    |CHAR
    |DOUBLE
    |BOOLEAN
    |STRING
;
Expresion
    : RESTA Expresion %prec UMENOS
    | NOT Expresion  {$$ = instruccionesAPI.OperacionBinaria($2,undefined,"!")}
    | Expresion SUMA Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"+")}
    | Expresion RESTA Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"-")}
    | Expresion MULTIPLICACION Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"*")}
    | Expresion DIVISION Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"/")}
    | Expresion MODULO Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"%")}
    | Expresion POTENCIA Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"^")}
    | Expresion AND Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"&&")}
    | Expresion OR Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"||")}
    | Expresion DOBLE_IGUAL Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"==")}
    | Expresion DIFERENTEA Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"!=")}
    | Expresion MENOR_IGUAL Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"<=")}
    | Expresion MENOR Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,"<")}
    | Expresion MAYOR_IGUAL Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,">=")}
    | Expresion MAYOR Expresion {$$ = instruccionesAPI.OperacionBinaria($1,$3,">")}
    | DECIMAL {$$ = {NUMERO_DEC:$1}}
    | NUMERO {$$ = {NUMERO:$1}}
    | TRUE   {$$ = {LOGICO:$1}}
    | FALSE {$$ = {LOGICO:$1}}
    | CADENA {$$ = {CADENA:$1}}
    | CARACTER {$$ = {CARACTER:$1}}
    | IDENTIFICADOR {$$ = {ID:$1}}
    | LlamarFuncion
    | PARENTESIS_APERTURA Expresion PARENTESIS_CIERRE
;
```

```
If
    :IF PARENTESIS_APERTURA Expresion PARENTESIS_CIERRE BLOQUE_INS //SOLO IF
        { $$ = instruccionesAPI.newIf( $3, $5, undefined, undefined) }
    |IF PARENTESIS_APERTURA Expresion PARENTESIS_CIERRE BLOQUE_INS Else//IF-ELSE
        { $$ = instruccionesAPI.newIf( $3, $5, undefined, $6) }
    |IF PARENTESIS_APERTURA Expresion PARENTESIS_CIERRE BLOQUE_INS ListELSEIF //con else if pero sin else
        { $$ = instruccionesAPI.newIf( $3, $5, $6, undefined) }
    |IF PARENTESIS_APERTURA Expresion PARENTESIS_CIERRE BLOQUE_INS ListELSEIF Else // if mas completo
        { $$ = instruccionesAPI.newIf( $3, $5, $6, $7 ) }
;
Else
    : ELSE BLOQUE_INS { $$ = instruccionesAPI.newElse($2) }
;
ListELSEIF //viene por lo menos un else if
    : ListELSEIF  Elseif {  $1.push($2); $$ = $1 }
    | Elseif  { $$ = [$1] }
;
Elseif
    : ELSE IF PARENTESIS_APERTURA Expresion PARENTESIS_CIERRE BLOQUE_INS
        { $$ = instruccionesAPI.newElseIf($4,$6) }
;
Switch
    : SWITCH PARENTESIS_APERTURA Expresion PARENTESIS_CIERRE LLAVE_APERTURA Lista_Case LLAVE_CIERRE
        { $$ =  instruccionesAPI.newSwitch($3, $6) }
;
Lista_Case
    : Lista_Case Case {  $1.push($2); $$ = $1 }
    | Case { $$ = [$1] }
;
Case
    : CASE Expresion DOS_PUNTOS BloqueCASES { $$ = instruccionesAPI.newCase($2,$4) }
    | DEFAULT DOS_PUNTOS BloqueCASES { $$ = instruccionesAPI.newCase("default",$3) }
;
Do
    : DO BLOQUE_INS WHILE PARENTESIS_APERTURA Expresion PARENTESIS_CIERRE PUNTO_COMA
        { $$ = instruccionesAPI.newDo_While($5,$2) }
```