TEC | Tecnológico de Costa Rica

Centro Académico de Alajuela
Escuela de Ingeniería en Computación
IC-5701 Compiladores e Intérpretes

Java Tropicalizado al Español
Analizador léxico

Avance #3

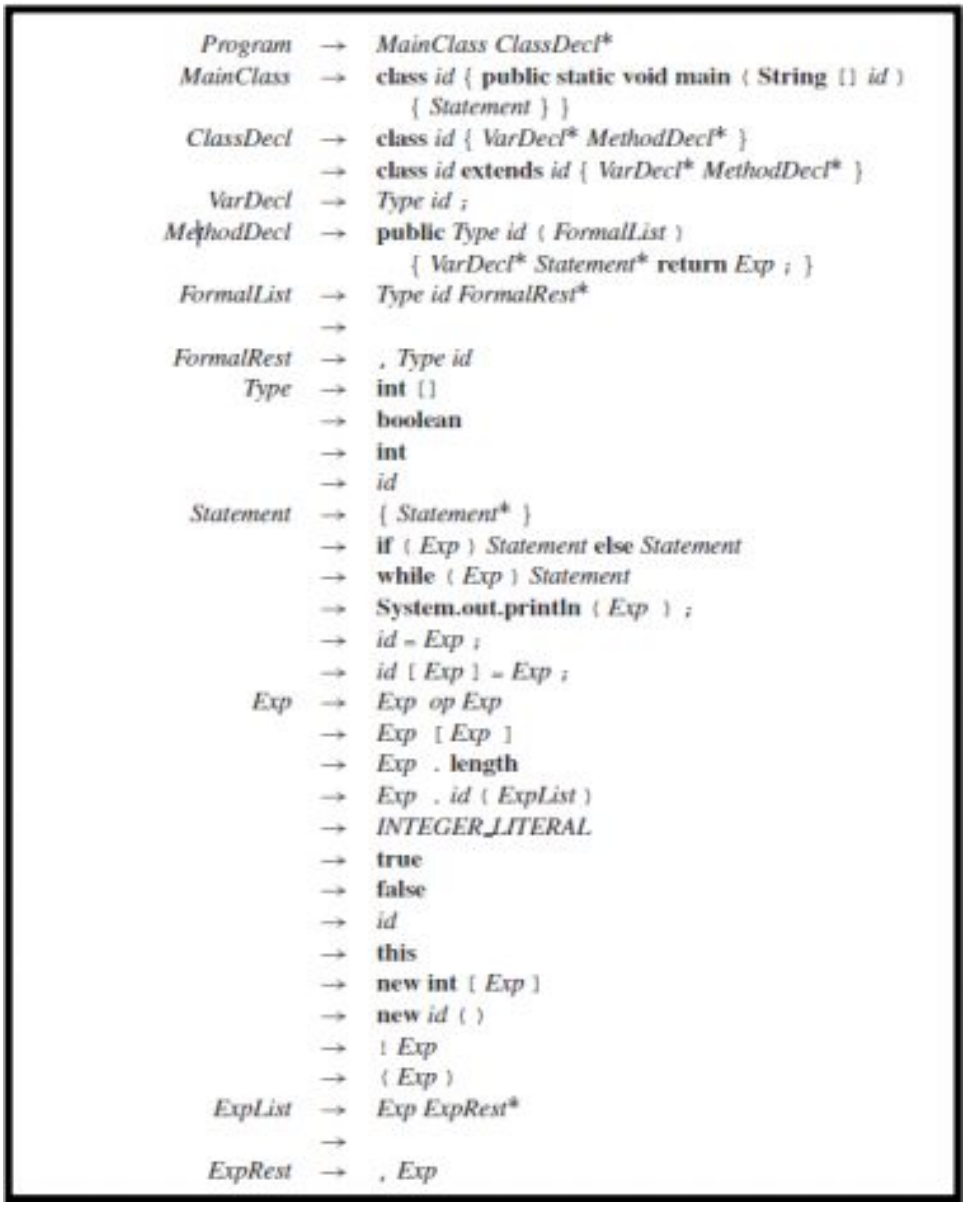
Alejandro Centeno Chaves
Hilary González Abarca
María Venegas Berrocal

Prof. Emmanuel Ramírez Segura
IS-2020

Mayo, 2020

Tabla de Contenidos

Gramática



Program → *MainClass ClassDecl**
MainClass → **class** *id* { **public static void main** (**String** [] *id*)
{ *Statement* } }
ClassDecl → **class** *id* { *VarDecl* MethodDecl** }
→ **class** *id* **extends** *id* { *VarDecl* MethodDecl** }
VarDecl → *Type id* ;
MethodDecl → **public** *Type id* (*FormalList*)
{ *VarDecl* Statement** **return** *Exp* ; }
FormalList → *Type id FormalRest**
→
FormalRest → , *Type id*
Type → **int** []
→ **boolean**
→ **int**
→ *id*
Statement → { *Statement** }
→ **if** (*Exp*) *Statement* **else** *Statement*
→ **while** (*Exp*) *Statement*
→ **System.out.println** (*Exp*) ;
→ *id* = *Exp* ;
→ *id* [*Exp*] = *Exp* ;
Exp → *Exp op Exp*
→ *Exp* [*Exp*]
→ *Exp* . **length**
→ *Exp* . *id* (*ExpList*)
→ *INTEGER_LITERAL*
→ **true**
→ **false**
→ *id*
→ **this**
→ **new int** [*Exp*]
→ **new** *id* ( )
→ ! *Exp*
→ (*Exp*)
ExpList → *Exp ExpRest**
→
ExpRest → , *Exp*

Programa	ClasePrincipal ClasDef*
ClasePrincipal	**clase** identificador { **publico estatico vacio principal** {**Cadena** [ ] identificador } { Declaracion } }
ClasDef	**clase** identificador { DeclVar* DeclMetodo* } **clase** identificador **extiende** identificador { DeclVar* DeclMetodo* }
DeclVar	Tipo identificador;
DeclMet	**publico** tipo identificador (ListaForm) { DeclVar* Declaracion* **retornar** Expresion ; }
ListaFormal	Tipo identificador RestoFormal
RestoFormal	, Tipo identificador
Tipo	**ent** [ ] **bool** **ent** identificador
Declaracion	{ Declaracion* } **si** (Expresion) Declaracion **entonces** Declaracion **mientras** (Expresion) Declaracion **Imprimir** (Expresion) ; identificador = Expresion ; ididentificador [Expresion] = Expresion ;
Expresion	Expresion operador Expresion Expresion [Expresion] Expresion .**largo** Expresion .identificador (ListaExp) ENTERO_LITERAL **Verdadero** **Falso** identificador **esto** **nuevo ent** [Expresion] **nuevo** identificador ( ) ! Expresion (Expresion)
ListaExp	Expresion RestoExp*
RestoExp	, Expresion

Reconocimiento de Tokens

EXPR. REGULAR TOKEN	TOKEN
[a-z][a-z0-9|"_"]*	Identificador
si, clase, mientras, entonces, imprimir, Verdadero, Falso, esto, nuevo, ent, publico, bool, extiende, retornar, largo, estatico, principal, Cadena	reservada
[0-9]+	entero
"s"	cadena
/ , + , * , - , & , | , == , != , < , > , <= , >=	operador
=	asignación
clase, extiende	ClasDef
Verdadero, Falso, esto, nuevo, !	expresion
ent, ent[ ], Cadena[ ],Cadena, bool, largo, vacio	Tipo
/*"s"*/	comentario
(	parentizq
)	parentder
{	corcheteizq
}	corcheteder

Logros, Errores y Problemas

- Se lograron reconocer efectivamente tokens para dígitos enteros y flotantes, cadenas, operadores, comentarios, etc.
- Tuvimos problemas al tratar de reconocer un string en lugar de un caracter porque no teníamos clara la forma para definirlo.
- Teníamos problemas al tratar de diferenciar entre un token y una palabra reservada ya que pensábamos que eran lo mismo.

- Tuvimos errores al querer abarcar más tokens de los necesarios.

  Código que será procesado por el analizador léxico:

```
clase Factorial {
        publico estatico vacio principal(Cadena[] a) {
                imprimir(nuevo Fac().HacerFac(10));
        }
}
clase Fac {
        publico ent HacerFac(ent num) {
                ent num_aux;
                si (num < 1)
                        num_aux = 1;
                entonces:
                        num_aux = num * (esto.HacerFac(num-1));
                        retornar num_aux;
        }
}
```

  Identificación de tokens para definición de clase e identificador del nombre de la clase, así como el reconocimiento de un paréntesis:

```
<ClasDef , clase>   <reservada , clase>   <Identificador , Factorial>   <corcheteizq , {>
```

  Identificación de tokens para el operador de asignación, de un identificador con un número entero:

```
<Identificador , num_aux>   <asignacion , =>   <entero , 1>
```

  Declaración de una variable de tipo numérico:

```
<Tipo , ent>   <Identificador , num_aux>
```

  Los demás tokens pueden verificarse utilizando el programa.