Construcción de un analizador léxico y sintáctico para un sublenguaje de Java

Iván Illán Barraya Javier Monescillo Buitrón

14 de marzo de 2019

Índice

1. Introducción

Java [1] es un lenguaje de programación de propósito general, concurrente y orientado a objetos que fue diseñado específicamente para tener las mínimas dependencias de implementación. Su intención principal es permitir que los desarrolladores de aplicaciones escriban el programa una única vez y lo ejecuten el cualquier dispositivo.

Lo que quiere decir que el código ejecutado en una plataforma no tiene que ser recompilado para correr en otra, así que se puede decir que es un lenguaje compilado e interpretado. Además Java es uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor en la web.

Esto último hace que sea el lenguaje perfecto para poder introducirlo en el problema que se quiere resolver.

2. Descripción del problema

El problema que se presenta es la construcción de un analizador léxico y sintáctico del lenguaje Java usando las herramientas Jflex y Cup [2].

El lenguaje que se propone es un sublenguaje de Java, en concreto una secuencia de métodos en Java que denotaremos como Jjava.

3. Solución propuesta

Para diseñar dichos analizadores, utilizaremos los conocimientos de la materia durante las distintas etapas del proceso.

- Diseño del sistema
- Diseño del Analizador Léxico
  - Identificar Tokens
  - Construcción del Analizador Léxico
- Diseño del Analizador Sintáctico
  - Especificación de la GLC
  - Creación del EBNF del lenguaje
  - Construcción de la gramática con Cup
- Diseño del Analizador Semántico

La primera etapa consta del diseño referente al lenguaje fuente o arquitectura general del sistema, incluido hasta el análisis léxico. Se proporcionará un diagrama total del problema.

Mientras que para la segunda etapa que será desarrollada en futuras entregas se tratará el análisis sintáctico y el análisis semántico.

4. Diseño del sistema

Para la estructura general del problema se proporciona una pequeña figura [3] donde se puede ver intuitivamente el funcionamiento del mismo.

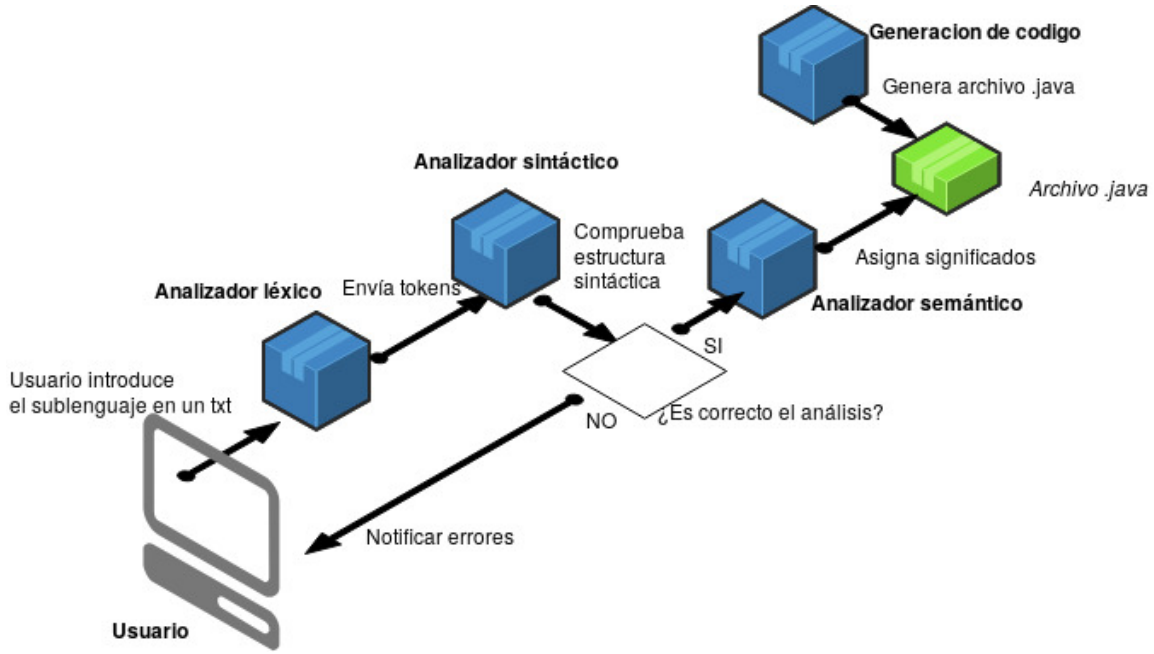

Diagrama general

Actualmente la parte de análisis semántico y generación de código no consta dentro del problema pero se incluye para futuras ampliaciones. Se limitará al mismo a la creación de la fase de análisis léxico y análisis sintáctico.

4.1. Lenguaje fuente

En esta tabla se pretende dar una descripción inicial de los elementos que tendrá el lenguaje.

Elementos del lenguaje	Ejemplo
Tipos de datos	int o boolean
Instrucción de asignación	type_var a = b; | type_var a = 0; | c = b;
Decremento	var–;
Incremento	var++;
Bucles	for(int i = 0; i<hola.length; i++) /* Ops*/
Llamadas a métodos	hola = calcularCosas(); | metodoVoid();
Retorno de valores	return x;
Secuencia de instrucciones	a = b; b * 2; //etc
Operaciones aritméticas	a + b; a - b; a * b; a / b;
Operaciones relacionales	a <b; a <= b; a >= b; a >b; a == b; a!=b;
Operaciones lógicas	a && b a | | b y !a
Cabecera de los métodos	public static
Tipo devuelto de un método	int, boolean, void

Notese, que para cerrar una sentencia es necesario de indicar al final de dicha sentencia el carácter ';' como se hace típicamente en Java.

4.2. Tabla de tokens

En la siguiente tabla de tokens, se muestran los lexemas de ejemplo, los tokens y las expresiones regulares asociadas a cada token.

Token	Lexema	Patrón
return	return	r·e·t·u·r· n
for	for	f·o·r
int	int	i·n·t
boolean	boolean	b·o·o·l·e·a·n
void	void	v·o·i·d
public static	public static	p·u·b·l·i·c· ·s·t·a·t·i·c
true	true	t·r·u·e
false	false	f·a·l·s·e
Asignación	=	=
Negación	!	!
Lógicos binarios	&&	&·& | |·|
Incrementos	++	+·+ | -·-
Relacionales	<	<| <·= | >| >·= | =·= | !·=
Paréntesis abierto	(	(
Paréntesis cerrado	)	)
Llave abierta	{	{
Llave cerrada	}	}
Punto y coma	;	;
Coma	,	,
Punto	.	.
Asterisco barra	*/	*·/
Barra asterisco	/*	/·*
ID	hola	[A-Za-z][A-Zaz0-9_]*
NUM	4	0 | [1-9][0-9]*
Valor nulo	null	n·u·l·l

5. Análisis léxico

El analizador léxico tiene varias funciones:

- Reconocer los símbolos que componen el texto fuente.
- Eliminar comentarios del texto fuente.
- Eliminar espacios en blanco, saltos de línea, tabulaciones, etc.
- Informar de los errores léxicos detectados

Como salida del analizador léxico, se obtiene una representación de la cadena de entrada en forma de cadena de **tokens**, que será posteriormente utilizada en la fase de análisis sintáctico. Para construir el analizador léxico se ha utilizado JFlex.

JFlex [4] es una herramienta software en la que se declaran los tokens que componen nuestro lenguaje fuente, así como las expresiones regulares asociadas a los mismos para poder generar el analizador léxico.

5.1. JFlex

En esta sección se muestra el código en JFlex donde se construye el analizador léxico del lenguaje Jjava.

Listing 1: Analizador Léxico en JFlex

```
package teoria_automatas;
import java.util.*;
import java.io.*;
import java_cup.runtime.Symbol;

%%

%class AnalizadorLexico
%unicode
%cup
%cupdebug

%line
%column

//Declaraciones

%{

private Symbol symbol(int type) {
return new Symbol(type, yyline, yycolumn);
}

private Symbol symbol(int type, Object value) {
return new Symbol(type, yyline, yycolumn, value);
}
%}

TERMINAR_LINEA = \r|\n|\r\n
CARACTERIN = [^\r\n]
ESPACIOBLANCO = {TERMINAR_LINEA} | [ \t\f]
COMENTARIO = {COMENTARIOT} | {FINLINEACOMENT}
COMENTARIOT = "/*" [^*] ~"*/" | "*/" "*"+ "/";
FINLINEACOMENT = "//" {CARACTERIN}* {TERMINAR_LINEA}?
ID = [a-zA-Z][a-zA-Z0-9"_""-"]*
LOGICOS_BINARIOS = "&&" | "||";
ARIT = "*" | "/" | "+" | "-";
RELACIONALES = "<" | "<=" | ">" | ">=" | "==" | "!=";
ASIGNACION = "="
INCREMENT = "++" | "--";
NUM = 0 | [1-9][0-9]*


%%
"null" {return symbol(sym.NULL, new String(yytext()));}
"return" {return symbol(sym.RETURN, new String(yytext()));}
"for" {return symbol(sym.FOR, new String(yytext()));}
"int" {return symbol(sym.INT, new String(yytext()));}
```

```
"boolean" {return symbol(sym.BOOLEAN, new String(yytext()));}
"void" {return symbol(sym.VOID, new String(yytext()));}
"public static" {return symbol(sym.BEGIN_METODOS, new String(yytext()));}
"true" {return symbol(sym.TRUE, new String(yytext()));}
"false" {return symbol(sym.FALSE, new String(yytext()));}
{ARIT} {return symbol(sym.ARIT, new String(yytext()));}
{RELACIONALES} {return symbol(sym.RELACIONALES, new String(yytext()));}
";" {return symbol(sym.PUNTOCOMA , new String(yytext()));}
{ASIGNACION} {return symbol(sym.ASIGNACION, new String(yytext()));}
{INCREMENT} {return symbol(sym.INCREMENT, new String(yytext()));}
{LOGICOS_BINARIOS} {return symbol(sym.LOGICOS_B, new String(yytext()));}
"!" {return symbol(sym.LOGICOS_U, new String(yytext()));}
"{" {return symbol(sym.LL_OP, new String(yytext()));}
"}" {return symbol(sym.LL_CL, new String(yytext()));}
"(" {return symbol(sym.PAR_OP, new String(yytext()));}
")" {return symbol(sym.PAR_CL, new String(yytext()));}
"," {return symbol(sym.COMA, new String(yytext()));}
"." {return symbol(sym.PUNTO, new String(yytext()));}
{ID} {return symbol(sym.ID, new String(yytext()));}
{COMENTARIO} {/*Ignoramos comentarios*/}
{ESPACIOBLANCO} {}
{NUM} {return symbol(sym.NUM, new String(yytext()));}

[^]  {System.out.println("Error lexico" +yytext());}
```

Referencias

[1] James Gosling, Bill Joy, Guy Steele, y Gilad Bracha, The Java language specification, tercera edición. Addison-Wesley, 2005. ISBN 0-321-24678-0.

[2] Java a Tope: Traductores Y Compiladores Con Lex/yacc, Jflex/cup Y Javacc, Rojas, Sergio Gálvez and Mata, Miguel Ángel Mora, 2005

[3] Draw.io, Alder Gaudenz, Benson David, En https://about.draw.io/ 2019

[4] Jflex, lexical analyzer generator for Java, Klein Gerwin, Rowe Steve, Décamps Régis, En https://www.jflex.de/, 2018