



# Paradigmas de Programación (EIF-400) Introducción Compilación con FP

CARLOS LORÍA-SÁENZ [LORACARLOS@GMAIL.COM](mailto:LORACARLOS@GMAIL.COM)

OCTUBRE 2017

EIF/UNA

# Objetivos

2

- ▶ Presentar nociones de compilación
- ▶ Estudiar el paradigma de FP junto con programación declarativa
- ▶ Conocer sobre generación automática de código (generación de parsers)
- ▶ Usar Java y Kotlin como casos de estudio

# Objetivos Específicos

3

- ▶ CFG, RE y ANTLR
- ▶ Patrones de visitadores y compilación
- ▶ Combinaciones OOP y FP y uso en compilación

# Material

4

- ▶ En el sitio
- ▶ Este material vale como demo para el Proyecto Programado II ([Kokoslan](#))

# Compilación

5

- ▶ Análisis Sintáctico (Parsing)
- ▶ Análisis Estático (Typing)
- ▶ Optimización y Generación
- ▶ Nos concentramos en Parsing y Generación en este material

- ▶ Lexer (o Tokenizer): reconocedor de unidades léxicas (tokens)
- ▶ Parser: reconocedor de unidades gramaticales (frases)
- ▶ Herramientas para especificar:
  - ▶ Lexer: Expresiones Regulares (RE) y Autómatas finitos (FA)
  - ▶ Parser: Gramáticas libres de contexto (CFG)
- ▶ Lexer produce stream de tokens
- ▶ Parser produce un AST (árbol de sintaxis abstracta, o árbol de parsing)



# Ejemplo

7

- ▶ Considere work/hello
- ▶ Especificación declarativa de un micro-parser
- ▶ Es el hola mundo de ANTLR4
- ▶ ANTLR4 generador de Parser a Java (hay otros puertos)

# ANTLR Preparación

8

- ▶ Usando la carpeta work/antlr
- ▶ Cópiala en C:
- ▶ Asegúrese de que su PATH y CLASSPATH apuntan a antlr\bats y antlr\bats\lib respectivamente

```
PP:set path=C:\antlr\bats;%PATH%
PP:set classpath=C:\antlr\lib\*;%CLASSPATH%
PP:antlr4
ANTLR Parser Generator  Version 4.5.3
-o _____ specify output directory where all output is generated
-lib _____ specify location of grammars, tokens files
-atn _____ generate rule augmented transition network diagrams
-encoding _____ specify grammar file encoding; e.g., euc-jp
-message-format _____ specify output style for messages in antlr, gnu, vs2005
-long-messages _____ show exception details when available for errors and warnings
-listener _____ generate parse tree listener (default)
-no-listener _____ don't generate parse tree listener
-visitor _____ generate parse tree visitor
-no-visitor _____ don't generate parse tree visitor (default)
-package _____ specify a package/namespace for the generated code
-depend _____ generate file dependencies
-D<option>=value _____ set/override a grammar-level option
-Werror _____ treat warnings as errors
-XdbgST _____ launch StringTemplate visualizer on generated code
-XdbgSTwait _____ wait for STviz to close before continuing
-Xforce-atn _____ use the ATN simulator for all predictions
-Xlog _____ dump lots of logging info to antlr-timestamp.log
```



# Estudie hello.g4

9

- ▶ Reglas de Lexer
- ▶ Reglas de Parser

Regla Parser

```
Hello.g4 x
1 // Define a grammar called Hello
2 grammar Hello;
3 r : 'hello' ID ; // match keyword hello followed by an identifier
4 ID : [a-z]+ ; // match lower-case identifiers
5 WS : [ \t\r\n]+ -> skip ; // skip spaces, tabs, newlines
6
```

Reglas  
de Lexer

# Hello

10

- Compilar gramática a java

```
03/10/2017 12:45 <DIR>
03/10/2017 12:45 <DIR>
29/09/2017 11:44 241 ..
                241 Hello.g4
                1 archivos 241 bytes
                2 dirs 19.065.749.504 bytes libres

PP:md java
PP:md classes
PP:antlr4 -o java Hello.g4
PP:javac -d classes java\*.java
PP:
```

Crear directorios de salidas

Compilar

# Hello...

11

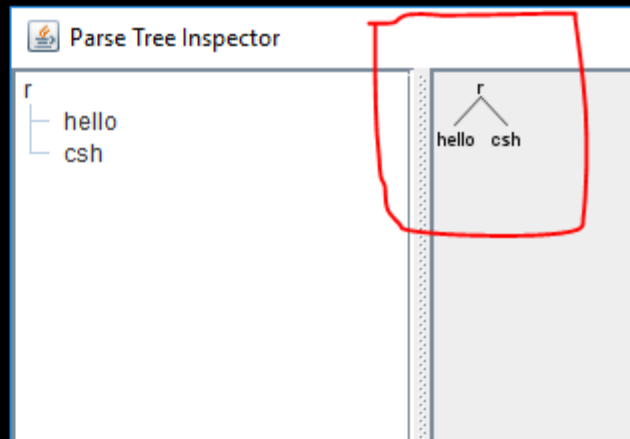
```
grammar Hello;  
r : 'hello' ID ;  
ID : [a-z]+ ;  
WS : [ \t\r\n]+ -> skip ;
```

```
PP:set classpath=classes;%classpath%
```

```
PP:grun Hello r -gui
```

```
hello csh
```

```
^Z
```



AST

# Ejercicio

12

- ▶ Permita mayúsculas en el ID

# Revise código generado

13

## ► Revise HelloParser.java

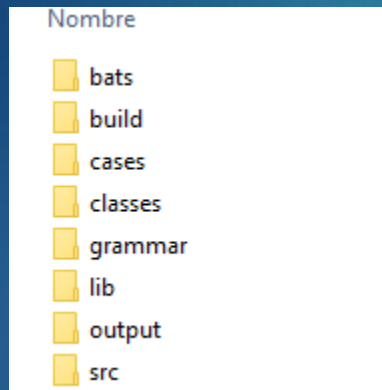
```
grammar Hello;  
r : 'hello' ID ;  
ID : [a-z]+ ;  
WS : [ \t\r\n]+ -> skip ;
```

```
public static class RContext extends ParserRuleContext {  
    public TerminalNode ID() { return getToken(HelloParser.ID, 0); }  
    public RContext(ParserRuleContext parent, int invokingState) {  
        super(parent, invokingState);  
    }  
    @Override public int getRuleIndex() { return RULE_r; }  
    @Override  
    public void enterRule(ParseTreeListener listener) {  
        if ( listener instanceof HelloListener ) ((HelloListener)listener).enterR(this);  
    }  
    @Override  
    public void exitRule(ParseTreeListener listener) {  
        if ( listener instanceof HelloListener ) ((HelloListener)listener).exitR(this);  
    }  
}
```

# Ejemplo Foo (pequeño proyecto)

14

- Estudie y pongo a correr Foo



```
// Foo = {V, P} V = NT U T
grammar Foo;
// NT = {a, b, t} (no terminales)
// T = {NUMBER, TRUE}

// P PARSER RULES (PRODUCTIONS)
a : b+      // ParserRule
;
b : d | t
;

d: NUMBER
;
t: TRUE
;
```



# Foo...

15

ANTLR

```
PP:set classpath=C:\antlr\lib\antlr-4.5.3-complete.jar
```

```
PP:bats\build_parser.bat
```

```
PP:bats\build_compiler.bat
```

```
PP:
```

Probamos con java

```
PP:bats\test_project.bat foo_case01.foo
```

```
Prueba el caso de prueba: cases\foo_case01.foo output\foo_case01.foo.out.foo
```

```
.....  
>>> FooBitc v0.0 CR EIF400.II-2017 <<<  
.....
```

```
Fooc Reading from cases\foo_case01.foo
```

```
Fooc Writing to output\foo_case01.foo.out.foo
```

```
PP:
```

# Foo... (probando Kotlin)

16

```
PP:bats\test_with_kotlin.bat foo_case01.foo
Prueba el caso de prueba: cases\foo_case01.foo output\foo_case01.foo.out.foo

.....
>>> FooBitc v0.0 CR EIF400.II-2017.kotlin <<<
.....

Fooc Reading from cases\foo_case01.foo
Fooc Writing to output\foo_case01.foo.out.foo

PP:
```

# Ciclo de Cambios

17

- ▶ Si cambia gramática: corra `build_parser` y `build_compiler`
- ▶ Si cambia `.java` o `.kt` solo corra `build_compiler`
- ▶ Si cambia casos de prueba sólo corra el test

# Modelo Visitor

18

- ▶ Cada regla de parser define un Contexto
- ▶ Regla `a : b+` define el contexto `AContext`
- ▶ Regla `b : d | t` define `BContext`
- ▶ Un `Context` da acceso a sus “hijos”
- ▶ `AContext` da acceso a los `b` (método que retorna `List`)
- ▶ `BContext` da acceso a `d` y `t`. Métodos para cada uno. Si alguno no viene es `null`

# Ejemplo

19

ANTLR - grun foo.compile.Foo a -gui

```
PP:set classpath=classes;%classpath%
PP:grun foo.compile.Foo a -gui
1 true true 2 3 4 true 5
^Z
```

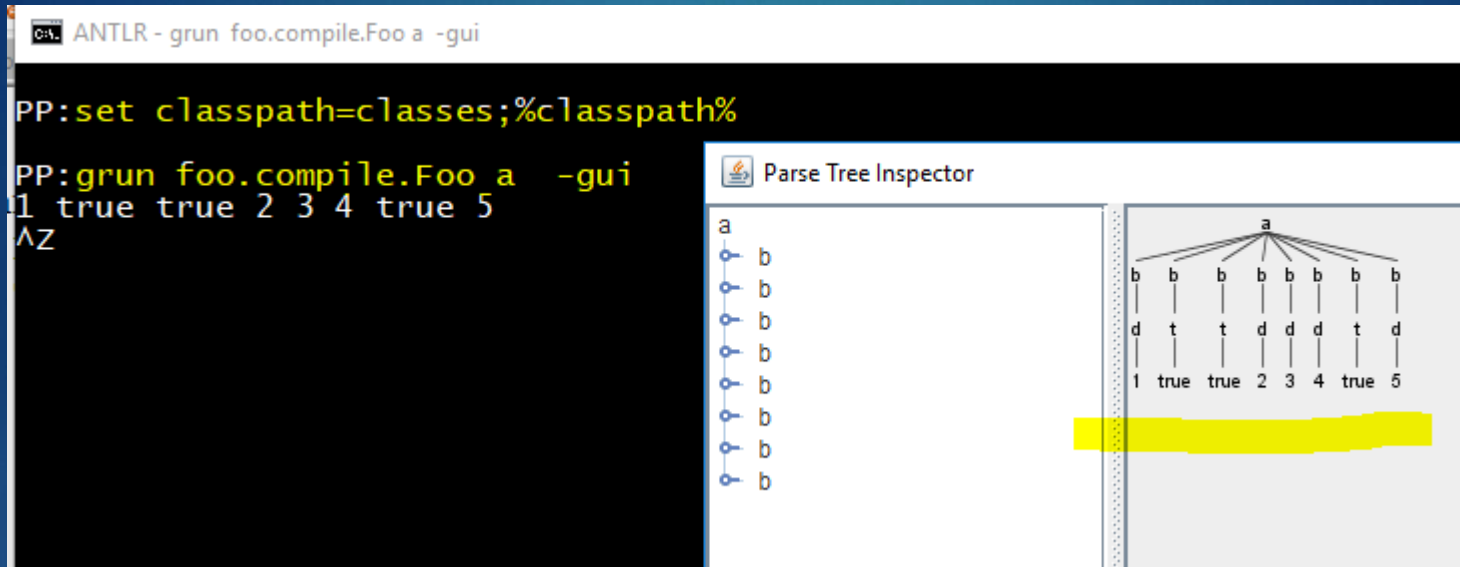
Parse Tree Inspector

a

- b
- b
- b
- b
- b
- b
- b
- b

a

- b
  - d
    - 1
- b
  - t
    - true
- b
  - t
    - true
- b
  - d
    - 2
- b
  - d
    - 3
- b
  - d
    - 4
- b
  - t
    - true
- b
  - d
    - 5



# Modelo Visitor

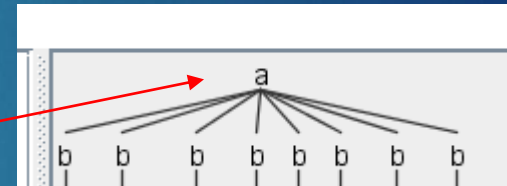
20

- Estudie FooParser y el modelo de visitor

```
// P PARSER RULES (PRODUCTIONS)
a : b+      // ParserRule
;
b : d | t
;
```

```
public static class AContext extends ParserRuleContext {
    public List<BContext> b() {
        return getRuleContexts(BContext.class);
    }
}
```

```
public static class BContext extends ParserRuleContext {
    public DContext d() {
        return getRuleContext(DContext.class, 0);
    }
    public TContext t() {
        return getRuleContext(TContext.class, 0);
    }
}
```





# Modelo Visitor...

21

```
d: NUMBER  
;  
t: TRUE  
;
```

d	t	t	d	d	d	t	d
1	true	true	2	3	4	true	5

```
public static class DContext extends ParserRuleContext {  
    public TerminalNode NUMBER() { return getToken(FooParser.NUMBER, 0); }  
}
```

```
public static class TContext extends ParserRuleContext {  
    public TerminalNode TRUE() { return getToken(FooParser.TRUE, 0); }  
    ...  
}
```

# Visitor y BaseVisitor

22

```
public interface FooVisitor<T> extends ParseTreeVisitor<T> {  
    /**  
    T visitA(FooParser.AContext ctx);  
    /**  
    T visitB(FooParser.BContext ctx);  
    /**  
    T visitD(FooParser.DContext ctx);  
    /**  
    T visitT(FooParser.TContext ctx);  
}
```

```
public class FooBaseVisitor<T> extends AbstractParseTreeVisitor<T> implements FooVisitor<T> {  
    /**  
    @Override public T visitA(FooParser.AContext ctx) { return visitChildren(ctx); }  
    /**  
    @Override public T visitB(FooParser.BContext ctx) { return visitChildren(ctx); }  
    /**  
    @Override public T visitD(FooParser.DContext ctx) { return visitChildren(ctx); }  
    /**  
    @Override public T visitT(FooParser.TContext ctx) { return visitChildren(ctx); }  
}
```

# Ejercicio

23

- ▶ Haga los cambios necesarios para que se parsee un caso de prueba como 1 true 2 false 4
- ▶ Construya un caso de prueba en cases y pruebe su solución