Redictado OO2

2024

Docentes:
Martin Tau
Leonardo Loza
German Ruiz
Felipe Dioguardi
Federico Balaguer
Alejandra Garrido

Redictado OO2: estrategias

Promoción

- Evaluación Refactoring (11 de Septiembre)
- Trabajo Final:
  - Con Evaluación Refactoring: Promoción
  - 10 Grupo de 2 alumnos
  - Presentación grupal: 27 de Noviembre
  - Entrega de código: Fechas de Parciales
  - Nota individual como final de materia

Cursada + Final

- Parcial (20/11, 04/12,18/12)
  - Template Method
  - Strategy
  - State
  - Composite
  - Façade
  - Adapter/Decorator
  - Command
  - Builder / Factory Method
  - Test Doubles
  - Refactoring de código
  - Frameworks
  - **Evaluación de casos (nuevo)**
- Final OO2

Fe de Errata y Aclaración

- Evaluación de Refactoring: 18 Septiembre 2024
- Cursada Promoción excluye parciales
  - La cursada se obtiene con el último commit en el git de la materia que pasa todos los tests

(deep into) Refactoring

Los refactoring de código son:

1. Cualquier transformación de código fuente
2. Cambios en el código que no alteran el comportamiento
3. Cambios en los datos del sistema para que funcione bien
4. Cambios en el código fuente que elimina code smells
5. Cambios en el código que mejoran el sistema
6. Transtornaciones del programador que arreglan el sistema
7. Cambios en el código que mantienen la funcionalidad
8. Mejoras en la funcionalidad sin cambios de código

Correctas: 2,7

Refactorings

- Rename Variable
- Rename Method
- Extract Method
- Move Method to Component
- Replace Conditional with Polymorphism
- Replace temp with query

Code Smells

- Duplicate Code
- Large Class
- Long Method
- Data Class
- Feature Envy
- Long Parameter List
- Switch Statements
- Primitive Obsession
- Oddball Solution
- Lazy Class

https://books.google.com.ar/books?id=Oxm8R1dY3RwC&newbks=1&newbks_redir=0&lpg=PA1&dq=refactoring%20to%20patterns&pg=PA1#v=onepage&q=refactoring%20to%20patterns&f=false

Patrones de Diseño

- Template Method
- Strategy
- State
- Composite
- Façade
- Adapter/Decorator
- Command
- Builder / Factory Method

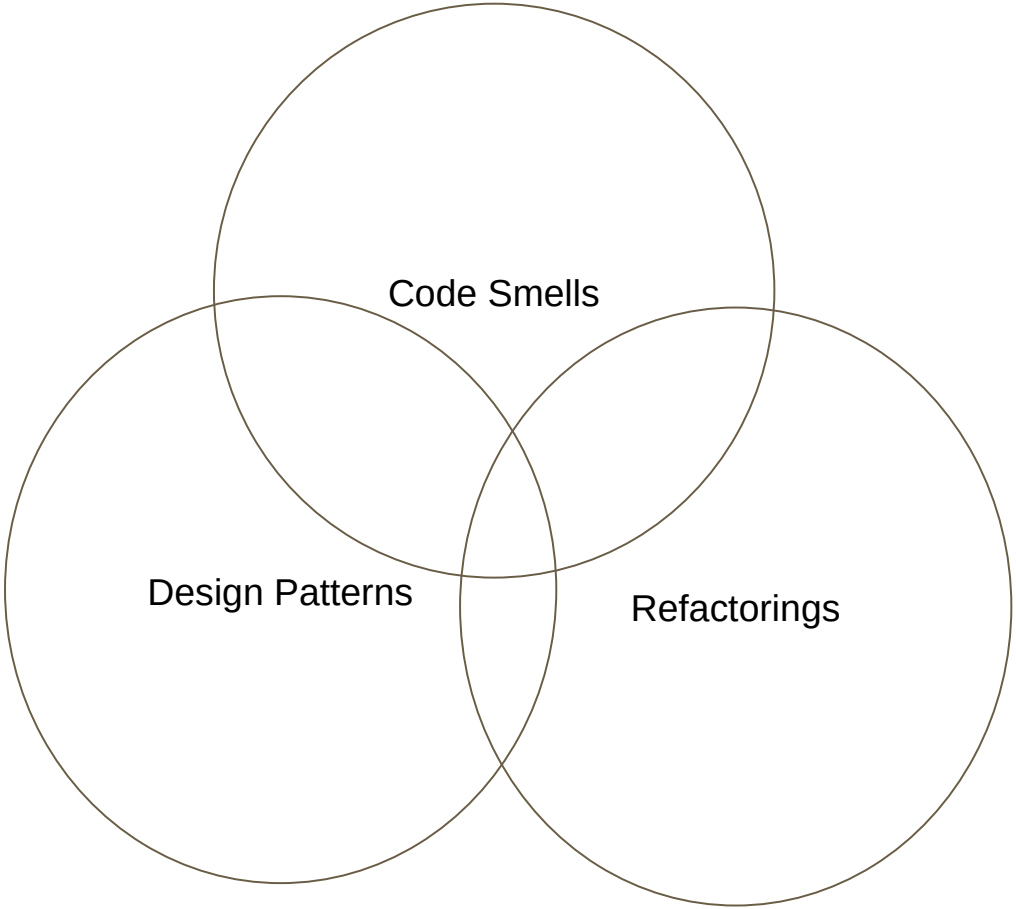
Code Smells → Refactoring (cuales se relacionan?)

- Duplicate Code
- Large Class
- Long Method
- Data Class
- Feature Envy
- Long Parameter List
- Switch Statements

- Rename Variable
- Rename Method
- Extract Method
- Move Method to Component
- Replace Conditional with Polymorphism
- Replace temp with query
- Variable move up
- Method move up (+ variables)

Refactorings →Patterns

- Form Template Method
- Extract Adapter
- Replace Implicit Tree with Composite
- Replace Conditional Logic with Strategy
- Replace State-altering conditionals with State


Code Smells
Design Patterns
Refactorings

Refactoring

- Transformaciones de código que preservan el comportamiento
- Preservar el comportamiento
  - (Hace lo que hacía antes) & (No hace lo que no hacía antes) *
- Se puede contar todas las maneras de escribir?
  - operación (función),
  - Clase
  - Librería, Framework
  - Programa
  - Elija su opción: Finito, Infinito contable o Infinito Incontable
- **Qué debería usarse para poder analizar (cualquier) código (dado un lenguaje de programación)?**

*Dificil… si requiere precisión terminamos en el universo de los métodos formales :-)

Qué debería usarse para poder analizar (cualquier) código ?

La herramienta secreta del informático...

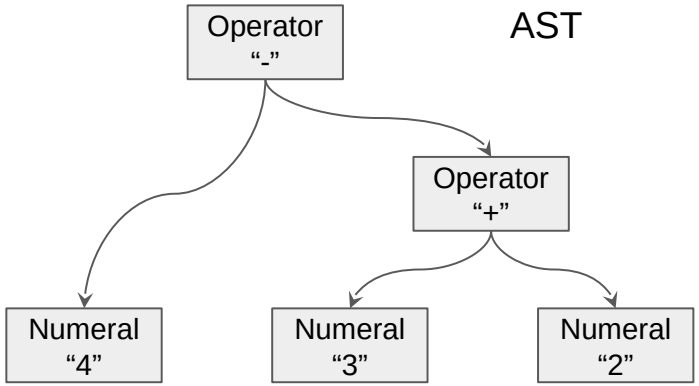
ABSTRACCIÓN

Qué debería usarse para poder analizar (cualquier) código ?

- **Abstracción**
  - Creada desde el código
  - Uniquivoca
  - Manipulable por Algoritmos
  - => Arboles !
- Syntax Trees
  - Parsing
  - Gramatica
  - Abstract (AST)
  - Concrete (CST)

EXPRESSION ::= NUMERAL | “(“NUMERAL”)”
“(“ EXPRESSION OPERATOR EXPRESSION “)”
OPERATOR ::= “+” | “-”
NUMERAL ::= DIGIT | DIGIT NUMERAL
DIGIT ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

Ejemplo: (4 - (3 + 2))

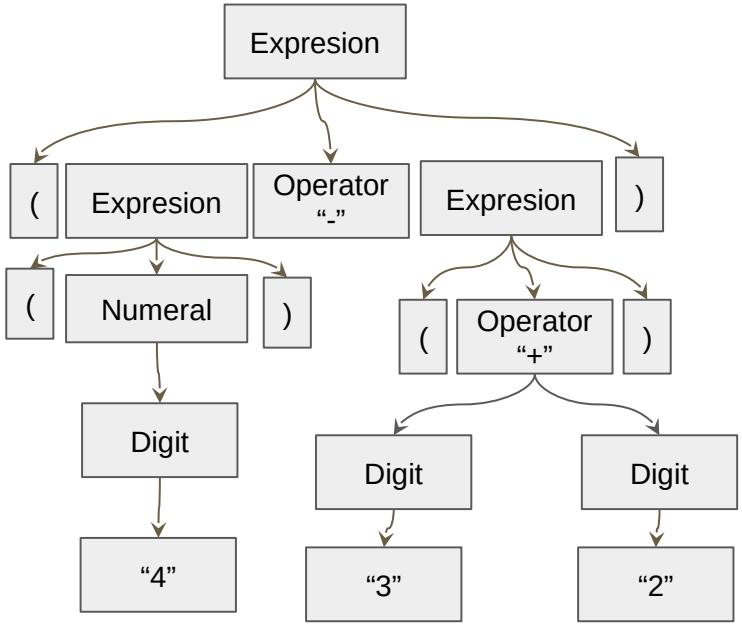


CST vs AST

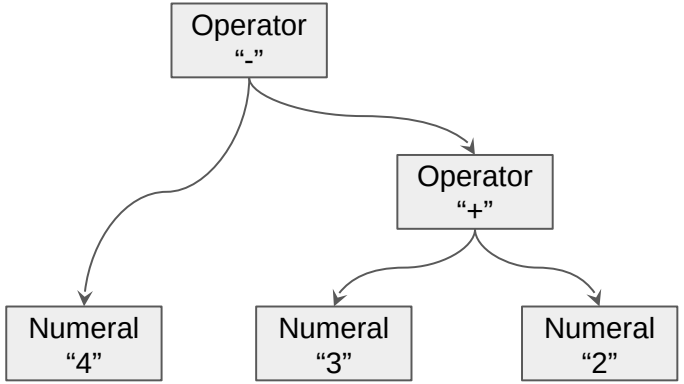
((4) - (3 + 2))

EXPRESSION ::= NUMERAL | “(“NUMERAL”)”
“(“ EXPRESSION OPERATOR EXPRESSION “)”
OPERATOR ::= “+” | “-”
NUMERAL ::= DIGIT | DIGIT NUMERAL
DIGIT ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

Concrete ST: cada derivation rule es un subarbol



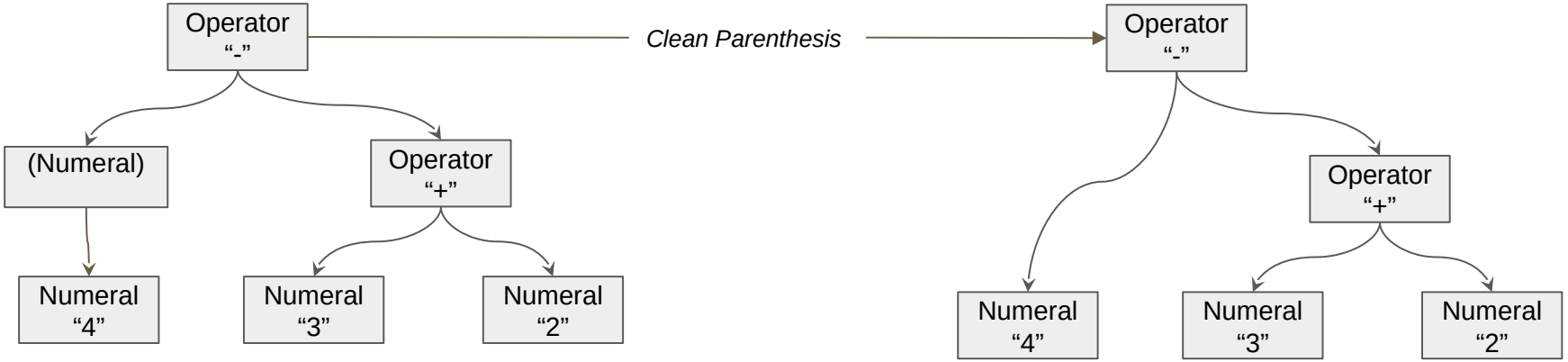
Abstract ST: estructura *relevante*



Refactoring(Tree)

EXPRESSION ::= NUMERAL | (NUMERAL)
“(“ EXPRESSION OPERATOR EXPRESSION “)”
(NUMERAL) ::= “(“ NUMERAL “)”
OPERATOR ::= “+” | “-”
NUMERAL ::= DIGIT | DIGIT NUMERAL
DIGIT ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

- Transformación(Tree) = Tree’
  - Tree EQUIV Tree’



Pretty print ((4) - (3 + 2))

Pretty print (4 - (3 + 2))

Refactorizar

- Transformaciones de AST que preservan la funcionalidad
- Son todas las expresiones equivalentes a ((4) - (3+2))? Cuales son las reglas de transformación que preservan funcionalidad?
  - **((4) - (3 + 2))**
  - (4 - (3 + 2))
  - 4 - (3 + 2)
  - (4) - (3 + 2)
  - (4) - 3 + 2
  - 4 - 3 + 2
  - 4 - 3 - 2

Para pensar...

Qué implica preservar el comportamiento?

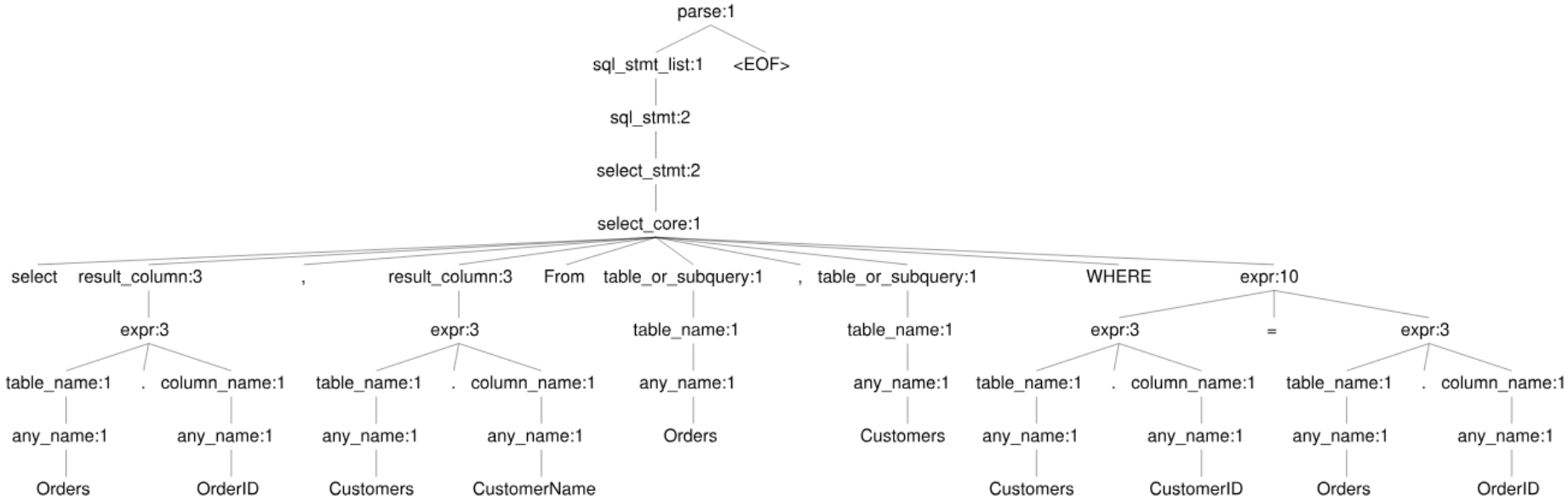
Pregunta de Examen…

1. Variable temporal (de una operación)
2. Variable de instancia privada de una clase
3. Variable de instancia publica de una clase
4. Método público de una clase
5. Método abstracto de una clase
6. Método público de una clase (pública) de una librería

Para cuando estan desvelados:

7. Semántica Denotacional
8. Lógica de Reescritura + Resolverdores de Teoremas

Select Orders.OrderID, Customers.CustomerName
From Orders, Customers
WHERE Customers.CustomerID = Orders.OrderID



Refactoring SQL: alias de tablas

```sql
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Customers, Orders
WHERE Customers.CustomerName='Around the Horn' AND Customers.CustomerID=Orders.CustomerID;
```

1. Verificar precondiciones
2. Agregar alias a la declaración de la tabla
3. Cambiar la ocurrencia de la tabla por su alias

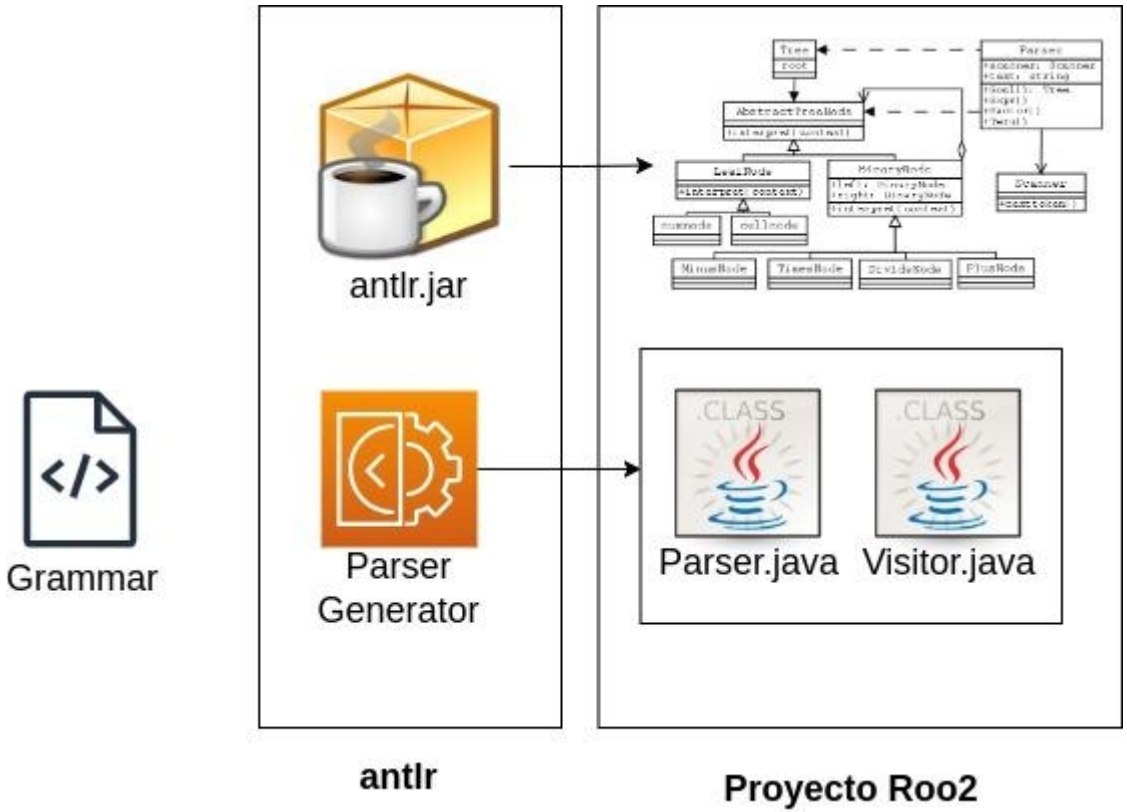
Ejemplo: Customers →"c". Orders →"o"

```sql
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

antlr

ANTLR (ANother Tool for Language Recognition)

- Generador de analizadores para leer, procesar, ejecutar o traducir texto estructurado o archivos binarios.
- Se usa ampliamente para crear lenguajes, herramientas y Frameworks.
- A partir de una gramática, ANTLR genera un analizador que puede construir y recorrer árboles de análisis.
- http://lab.antlr.org/


antlr.jar
Grammar
Parser
Generator
.CLASS
.CLASS
Parser.java
Visitor.java
antlr
Proyecto Roo2

Antlr lab: herramiento de pruebas

Lexer Parser Sample

```
parser grammar ExprParser;
options { tokenVocab=ExprLexer; }

program
    : stat EOF
    | def EOF
    ;

stat: ID '=' expr ';'
    | expr ';'
    ;

def : ID '(' ID (',' ID)* ')' '{' stat* '}' ;

expr: ID
    | INT
    | func
    | 'not' expr
    | expr 'and' expr
    | expr 'or' expr
    | expr SUM expr
    | expr SUB expr
    | expr MUL expr
    | expr DIV expr
    | expr POW expr
    ;

func : ID '(' expr (',' expr)* ')' ;
```

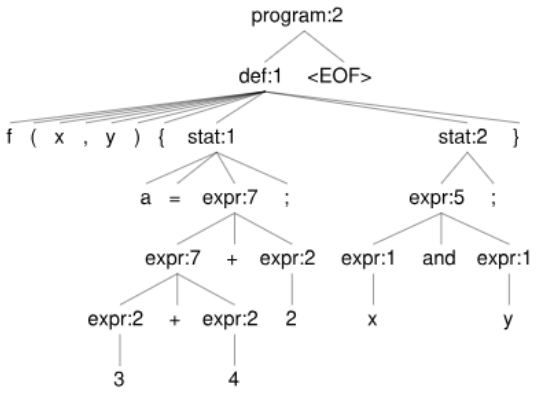
Input sample.expr

```
f(x,y) {
    a = 3+4+2;
    x and y;
}
```

Start rule

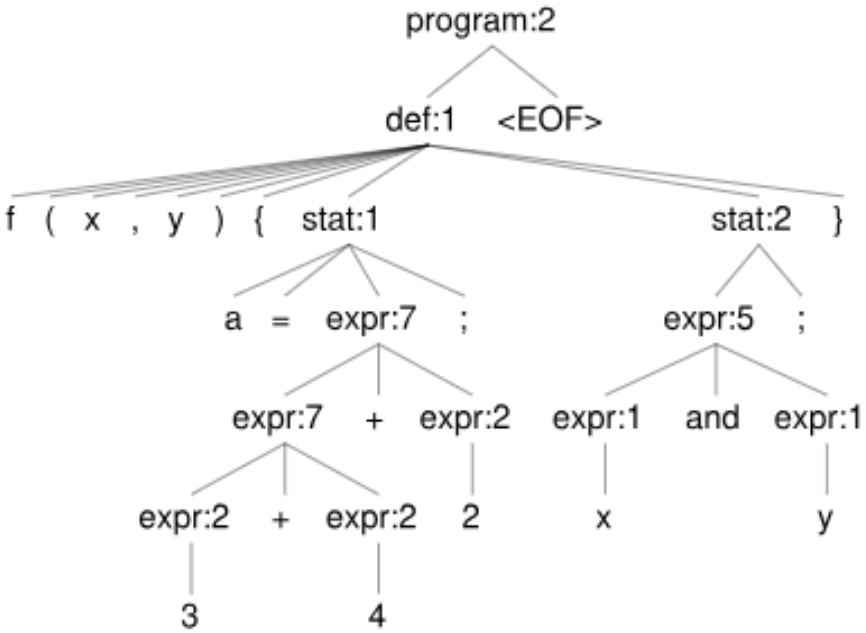
program Run Show profiler

Tree Hierarchy



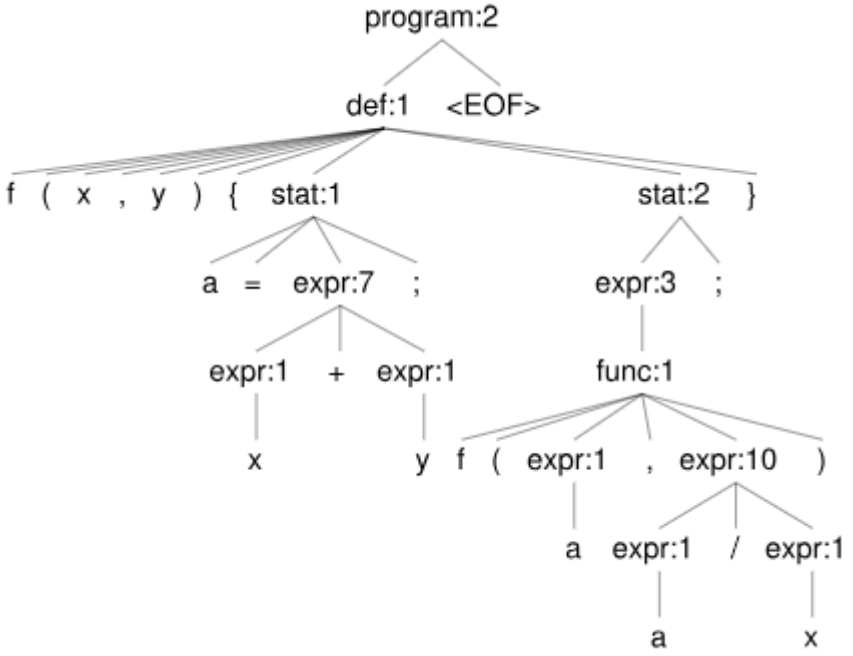
```
 parser grammar ExprParser;
options { tokenVocab=ExprLexer; }

program
    : stat EOF
    | def EOF
    ;

stat: ID '=' expr ';'
    | expr ';'
    ;

def : ID '(' ID (',' ID)* ')' '{' stat* '}' ;

expr: ID
    | INT
    | func
    | 'not' expr
    | expr 'and' expr
    | expr 'or' expr
    | expr SUM expr
    | expr SUB expr
    | expr MUL expr
    | expr DIV expr
    | expr POW expr
    ;

func : ID '(' expr (',' expr)* ')' ;
```

```
f(x,y) {
    a = 3+4+2;
    x and y;
}
```



```
 parser grammar ExprParser;
options { tokenVocab=ExprLexer; }

program
    : stat EOF
    | def EOF
    ;

stat: ID '=' expr ';'
    | expr ';'
    ;

def : ID '(' ID (',' ID)* ')' '{' stat* '}' ;

expr: ID
    | INT
    | func
    | 'not' expr
    | expr 'and' expr
    | expr 'or' expr
    | expr SUM expr
    | expr SUB expr
    | expr MUL expr
    | expr DIV expr
    | expr POW expr
    ;

func : ID '(' expr (',' expr)* ')' ;
```

```
1  f(x,y) {
2      a = x + y;
3      f(a, a/x);
4  }
5
```



roo2.next()

- Code Smells del lenguaje Sample (disponible en antlr lab)
  - Version extendida por Roo2 (ver moodle)
- Refactoring
  - OO2 2024
  - Sample en base a los CST