SOM2IF15 – Compilation

From ANTLR Parse Trees to Abstract Syntax Trees

Frédéric Loulergue

2022

The Problem

- ANTLR generates a set of class to represent parse trees
- In the rest of the compiler, we want to manipulate abstract syntax trees (AST), and we implemented a hierarchy of classes to implement such trees (in package `ast`)

$\Longrightarrow$ We need to transform parse trees to AST

The Solution

Implement a visitor for parse trees to build abstract syntax trees

ANTLR

Reminder

- Option `-visitor` to generate code to support the Visitor Pattern
- For a grammar `G` generates two files:
  - `GVisitor.java` (interface)
  - `GBaseVisitor.java` (class)

ANTLR Parse Trees

- The nodes are object of sub-classes of `Context`
- If the rules for a given non-terminal are not labelled, the generated name is: the non-terminal + `Context` and there is *one class per non-terminal*
- If the rules are labelled, there is *one generated class per rule* and the class are named: label + `Context`

Examples

Grammar

```
grammar pico;
pico: (instruction ';')* EOF                #Program
    ;
```

Generated Class for the Parse Tree Node

```
public static class ProgramContext {
  public TerminalNode EOF()
  { /* ... */ }
  public List<InstructionContext> instruction()
  { /* ... */ }
  public InstructionContext instruction(int i)
  { /* ... */ }
  /* ... */
}
```

Example

Grammar

```
expression :  expression op=(MUL|DIV) expression #MulDiv
              // ...
MUL : '*';
DIV : '/';
```

Generated Class for the Parse Tree Node

```
public static class MulDivContext {
   public Token op;
   public List<ExpressionContext> expression()
   { /*...*/ }
   public ExpressionContext expression(int i)
   { /*...*/ }
   // ...
}
```

ANTLR Visitor

Principle

For each non-terminal or label `L`:

```
public T visitL(LContext ctx)
```

Generated `picoVisitor`

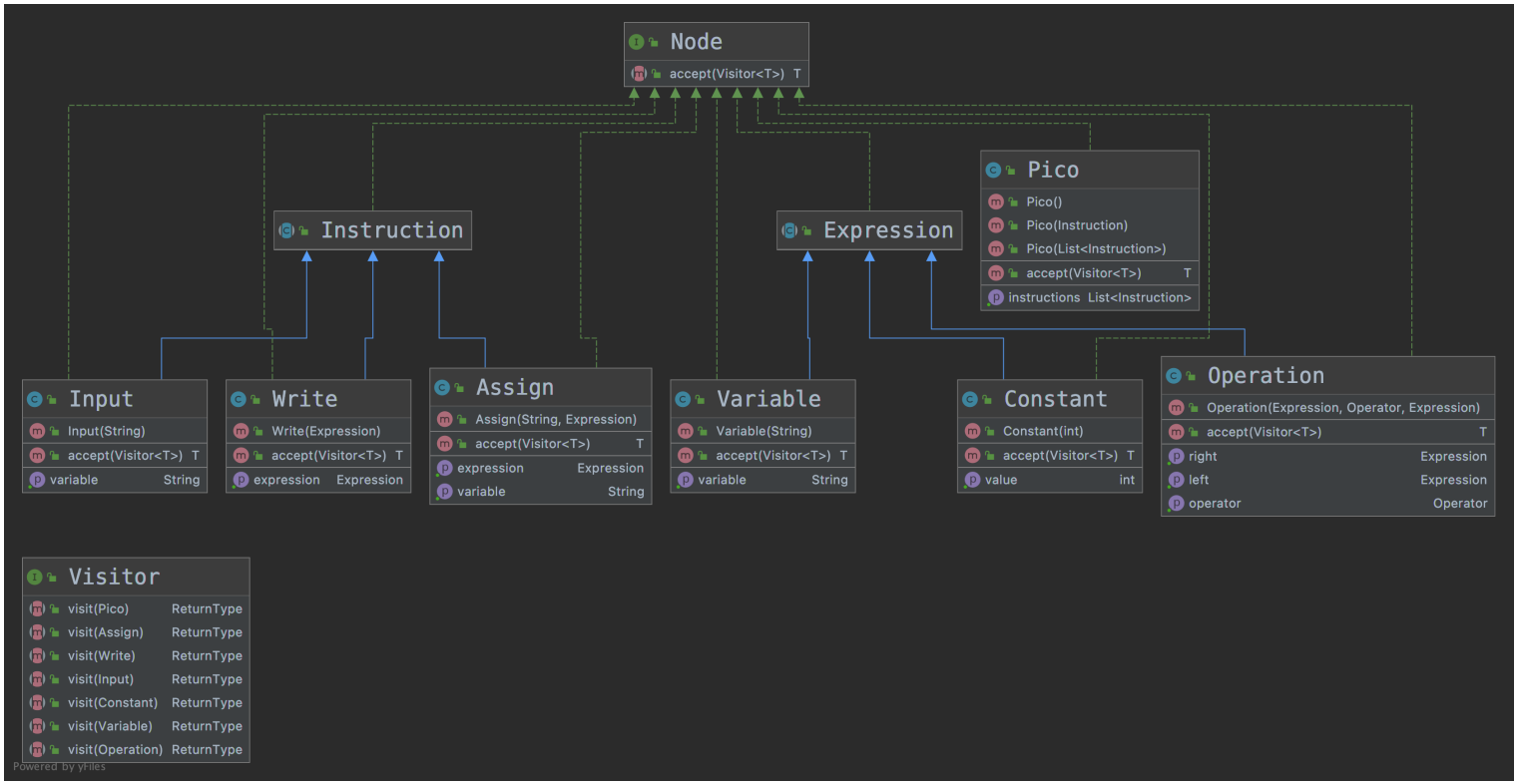
```
public interface picoVisitor<T> {
  T visitProgram(picoParser.ProgramContext ctx);
  // ...
  T visitMulDiv(picoParser.MulDivContext ctx);
  // ...
}
```

From Parse Tree to AST

Principle

Implement a visitor for the ANTLR visitor

Reminder: `pico`'s AST



From Parse Tree to AST: pico Example 1

```
public class Build extends picoBaseVisitor<Node> {

  @Override
  public Node visitProgram(picoParser.ProgramContext ctx) {
    List<Instruction> instructionList = new ArrayList<>();
    for(picoParser.InstructionContext insCntxt : ctx.instruction()){
      Instruction ins = (Instruction) insCntxt.accept(this);
      instructionList.add(ins);
    }
    return new Pico(instructionList);
  }
```

From Parse Tree to AST: pico Example 2

```
private Operator operatorFromToken(Token op) {
  switch (op.getText()) {
    case "+": return Operator.ADD;
    case "-": return Operator.SUB;
    case "*": return Operator.MUL;
    case "/": return Operator.DIV;
  }
  throw new Error("Unknown operator in parse tree.");
}
@Override
public Node visitMulDiv(picoParser.MulDivContext ctx) {
  Expression left = (Expression) ctx.expression(0).accept(this);
  Expression right = (Expression) ctx.expression(1).accept(this);
  return new Operation(left, operatorFromToken(ctx.op), right);
}
```

- `getType` on an object of type `Token` returns one of the lexer's constants
- `picoLexer.tokens` contains the list of lexer types
- `getText()` returns the string related to the token

Taking positions in the source text into account

Error Message of our Compiler

- To be more user friendly, each error should be located precisely in the source code
- This means our AST nodes should contain position information

In the `ast` package

- The class `Node` has been modified
- You need to modify all the constructors in of your AST classes to deal with the position information
- CONVENTION: the position argument should be the first one

ast/Position.java

```
public class Position {
  private int line;
  private int column;
  public Position(int line, int column) {
    this.line = line;
    this.column = column;
  }
  @Override
  public String toString() {
    return "[line=" + line + ", column=" + column + "]";
  }
}
```

New version of ast/Node.java

```
public abstract class Node {
    private Position position;
    public Position getPosition() {
        return position;
    }
    abstract <T> T accept(Visitor<T> visitor);
}
```

Example of a Modified Class

```
package ast;

public class ExpVariable extends Expression {

    private final String variable;

    public ExpVariable(Position position, String variable) {
        this.position = position;
        this.variable = variable;
    }

    public String getVariable() {
        return this.variable;
    }

    @Override
    <T> T accept(Visitor<T> visitor) {
        return visitor.visit(this);
    }
}
```

How to Access the Position in ANTLR Code?

```
// Building an AST Position from an ANTLR Context
private static
   Position position(ParserRuleContext ctx) {
     return new
       Position(ctx.start.getLine(),
                ctx.start.getCharPositionInLine());
}
```