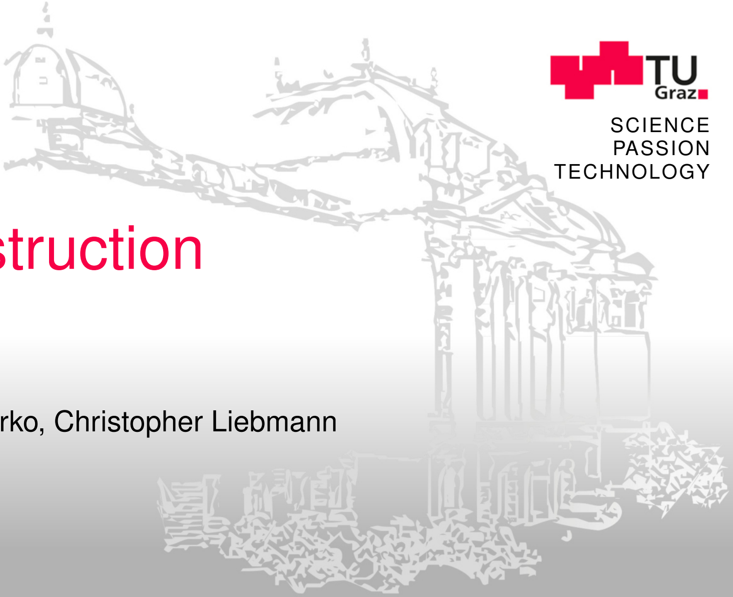
SCIENCE
PASSION
TECHNOLOGY



Compiler Construction
Task 1

Sebastian Puck, Alexander Perko, Christopher Liebmann

March 7, 2024

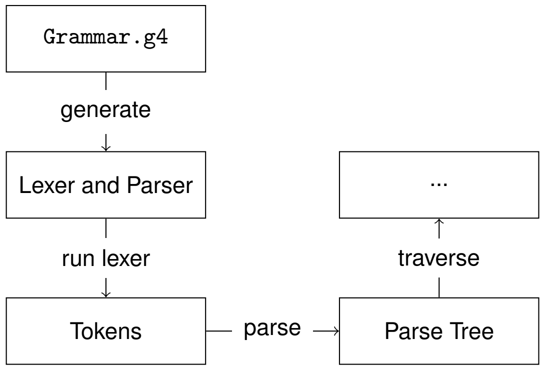
Outline

Why Lexers and Parsers

- we want to inspect some input ...
- ... to verify if it follows a grammar

ANother Tool for Language Recognition (ANTLR)

1. define a grammar
2. generate lexer and parser
3. retrieve parse tree
4. traverse parse tree



ANTLR 4 - Lexer and Parser Rules

- lexer rules start in uppercase
- parser rules start in lowercase
- rules defined at the top are considered first

```
// lexer rules
START : '[' ;
END : ']' ;
ID : [A-Za-z]+ ;
VALUE_INT : DIGIT+ ;
fragment DIGIT : [0-9] ;

// parser rules
attribute : ID ':' VALUE_INT ';' ;
object : START attribute+ END EOF ;
```

ANTLR 4 - Left Recursion

- direct left recursion is supported (line 4)
- indirect left recursion is not supported (line 5)
- rewrite indirect to direct left recursion

```
1  AND : '&' ;
2  OR : '|' ;
3
4  operand : operand AND operand
5          | otherOperation
6          ;
7
8  otherOperation : operand OR operand ;
```

ANTLR 4 - Alternative Labels

- parser rules support alternatives
- `value` is either `VALUE_INT` or `VALUE_STRING`
- `#<label>` to label each alternative

```
value : VALUE_INT       # INT
      | VALUE_STRING    # STRING
      ;
```

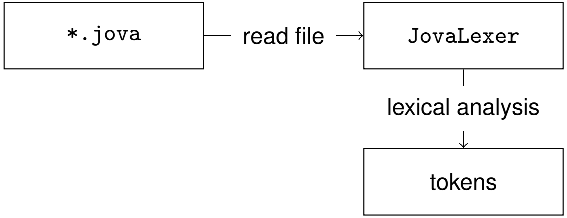
- label as orientation for generated visitor / listener methods
- for example: `visitINT`, `visitSTRING`

Live Demo

ANTLR 4 Lexer and Parser

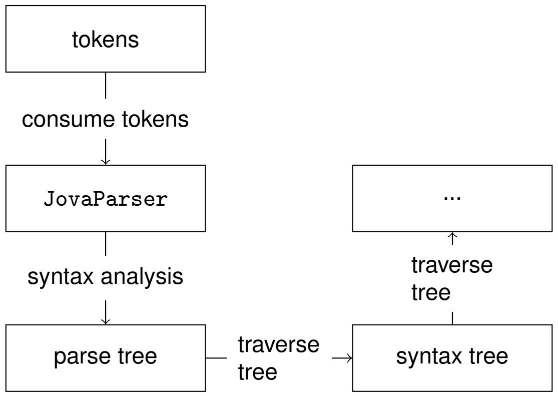
Jova Compiler - Lexer

- process `*.jova` files
- perform lexical analysis
- generate tokens


$$\begin{tikzcd}
\texttt{*.jova} \arrow[r, "\text{read file}"] & \texttt{JovaLexer} \arrow[d, "\text{lexical analysis}"] \\
 & \text{tokens}
\end{tikzcd}$$

Jova Compiler - Parser

- run syntax analysis
- produce parse tree
- further operations
  - build data structures
  - inspect other attributes



Gradle Tasks

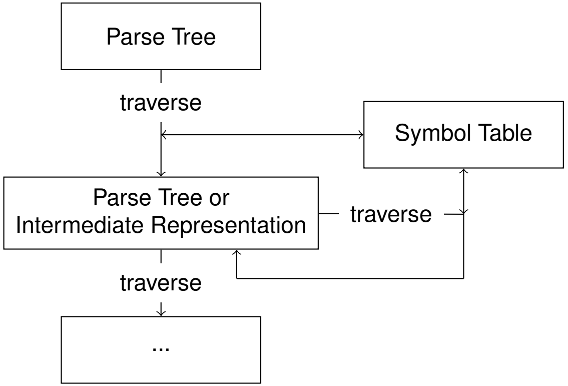
- `./gradlew compileJava`
  compiles the project
- `./gradlew compileTestJava`
  compiles the project with test files
- `./gradlew testRig -P fileName=<jova_file>`
  visualizes the parse tree
- `./gradlew clean`
  cleans the project
- `./gradlew test`
  executes tests

Why semantic analysis

- parsing checks an input's syntax
  - does the input follow a grammar?
- semantic analysis performs further checks
  - checking types
  - validating class inheritance
  - and more

Data Structures and how to use them

1. perform semantic analysis
   - traverse and build trees
   - read / write symbol table
   - handle problems
2. further steps (e.g. code generation)



ANTLR 4 - Traversing Parse Trees

Visitors

- can use java method returns
- ANTLR 4 generates `<grammar>BaseVisitor` class

Listeners

- no method returns, have to simulate them
- ANTLR 4 generates `<grammar>BaseListener` class

ANTLR 4 - Error Handling

- errors may occur when
  - executing the lexer or
  - running the parser
- default error handling can be altered
  - extend ANTLR's `BaseErrorListener` class

Symbol Table

- data structure involved in several compile steps
- matches the language's requirements
  - classes
  - variables
  - environments
- helps to perform checks

Live Demo

ANTLR 4 Parse Tree Traversal

Jova - Basics

- types (`int`, `bool`, `string`)
- class definitions and `nix` type
- built-in functions (`print`, `readInt`, `readLine`)
- operators, conditions and return statements
- main method

Jova - Environments

- type of environments
  - program
  - class
  - method
- if and while statements do not have environments
- identifiers and method signatures are unique within an environment
- field shadowing

Jova - Inheritance

- a class can inherit from another class
- a subclass contains the fields of its superclass
- field and method hiding
- method overwriting

Live Demo

Task 1 Framework and Assignment Sheet