# ANTLR Cheatsheet for `Prev25Lexer.g4` and `Prev25Parser.g4`

## Introduction

ANTLR (Another Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. This cheatsheet covers common constructs and patterns used in `Prev25Lexer.g4` and `Prev25Parser.g4`.

## Lexer Rules

### Definition

Lexer rules define how to break input into tokens. Lexer rule names are written in all uppercase.

```
ID : [a-zA-Z_] [a-zA-Z_0-9]* ;
NUMBER : [0-9]+ ;
```

### Operators and Constructs

- `[a-z]` – Matches any character in the range
- `.` – Matches any character (except newline, unless specified)
- `+` – One or more repetitions
- `*` – Zero or more repetitions
- `?` – Zero or one (optional)
- `'` or `"` – Matches literal characters/strings

### Lexer Utilities

- `fragment` – Defines a reusable part of a lexer rule
- `mode` – Switch between different lexical modes
- `channel(HIDDEN)` – Sends tokens to a hidden channel (e.g. whitespace, comments)
- `-> skip` – Tells lexer to discard the token

### Example

```
fragment DIGIT : [0-9] ;
WS : [ \t\r\n]+ -> skip ;
COMMENT : '//' ~[\r\n]* -> skip ;
```

# Parser Rules

## Definition

Parser rules define the grammar's syntax and how tokens are combined into structures. Rule names are lowercase.

```
expr : term (('+' | '-') term)* ;
term : factor (('*' | '/') factor)* ;
```

## Operators and Constructs

- `|` – Choice (alternatives)
- `?` – Optional (zero or one)
- `*` – Zero or more
- `+` – One or more
- `-> label` – Assigns a label or specifies a rule transformation
- `#Label` – Alternative labeling for parse tree listeners/visitors

## Predicates and Semantic Checks

```
rule : {isEnabled}? ID ;
```

**Semantic predicates** are Java expressions in braces followed by a question mark. They control rule matching based on runtime conditions.

## Actions: Embedding Java Code

Actions allow embedding target language code (usually Java) within rules or grammar-level blocks.

- `@header {}` – Insert code at the top of the generated file
- `@members {}` – Insert member variables or methods into parser/lexer class
- `@init {}` – Run code at the start of a rule
- Rule-local: { ... } inside rules

## Example with Actions

```
@header {
import java.util.*;
}

@members {
int exprCount = 0;
}

expr
@init { exprCount++; }
    : left=term op=('+'|'-') right=term {
        System.out.println("Expr #" + exprCount + ": " + $left.text + " " + $op.text + "
             " + $right.text);
    }
    ;
```

## ANTLR Usage Workflow

1. Define lexer and parser grammar files.
2. Run: `antlr4 MyGrammar.g4`
3. Compile: `javac *.java`
4. Test: `grun MyGrammar startRule -tokens` or `-gui`

## Best Practices

- Keep lexer and parser rules in separate files for clarity.
- Avoid embedding too much Java logic in grammar — delegate to listeners/visitors.
- Use fragments to eliminate repetition in lexer patterns.
- Use labeled alternatives and rule names to simplify tree traversal.