ANTLR features

ANTLR – ANother Tool for Language Recognition

```
expr: expr '*' expr
    | expr '+' expr
    | id
    ;
```

Parser rules: EBNF

```
//PARSER RULES
id : ID;
program: expr EOF;
expr: expr '*' expr
    | expr '+' expr
    | id
    ;
```

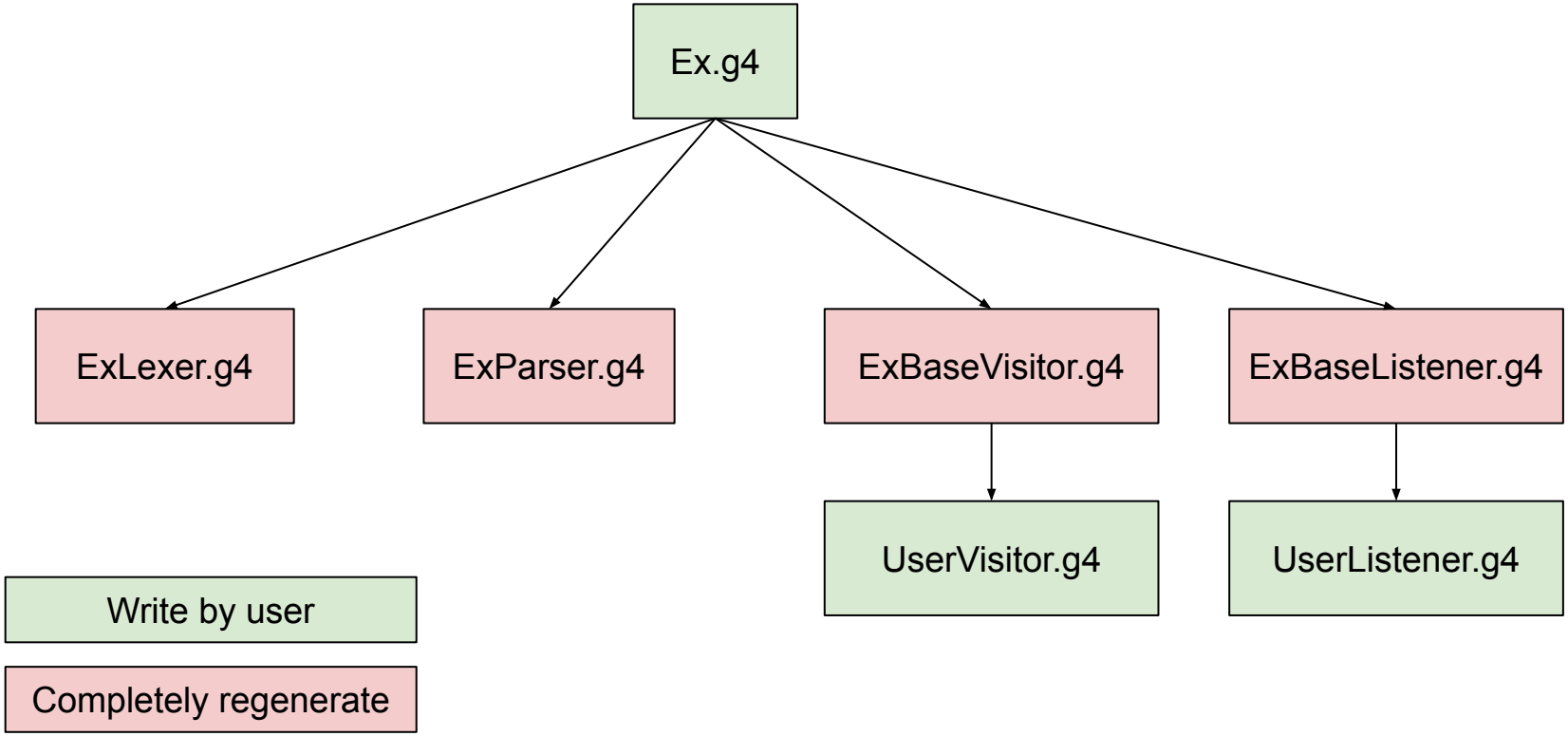
Lexerless: lexer and parser rules in one file

```
//PARSER RULES
id : ID;
program: expr EOF;
expr: expr '*' expr
    | expr '+' expr
    | id
    ;

//LEXER RULES
ID : [A-Za-z]+ ;              // match id with upper, lowercase
WS : [ \t\r\n]+ -> skip ;     // ignore whitespace
```

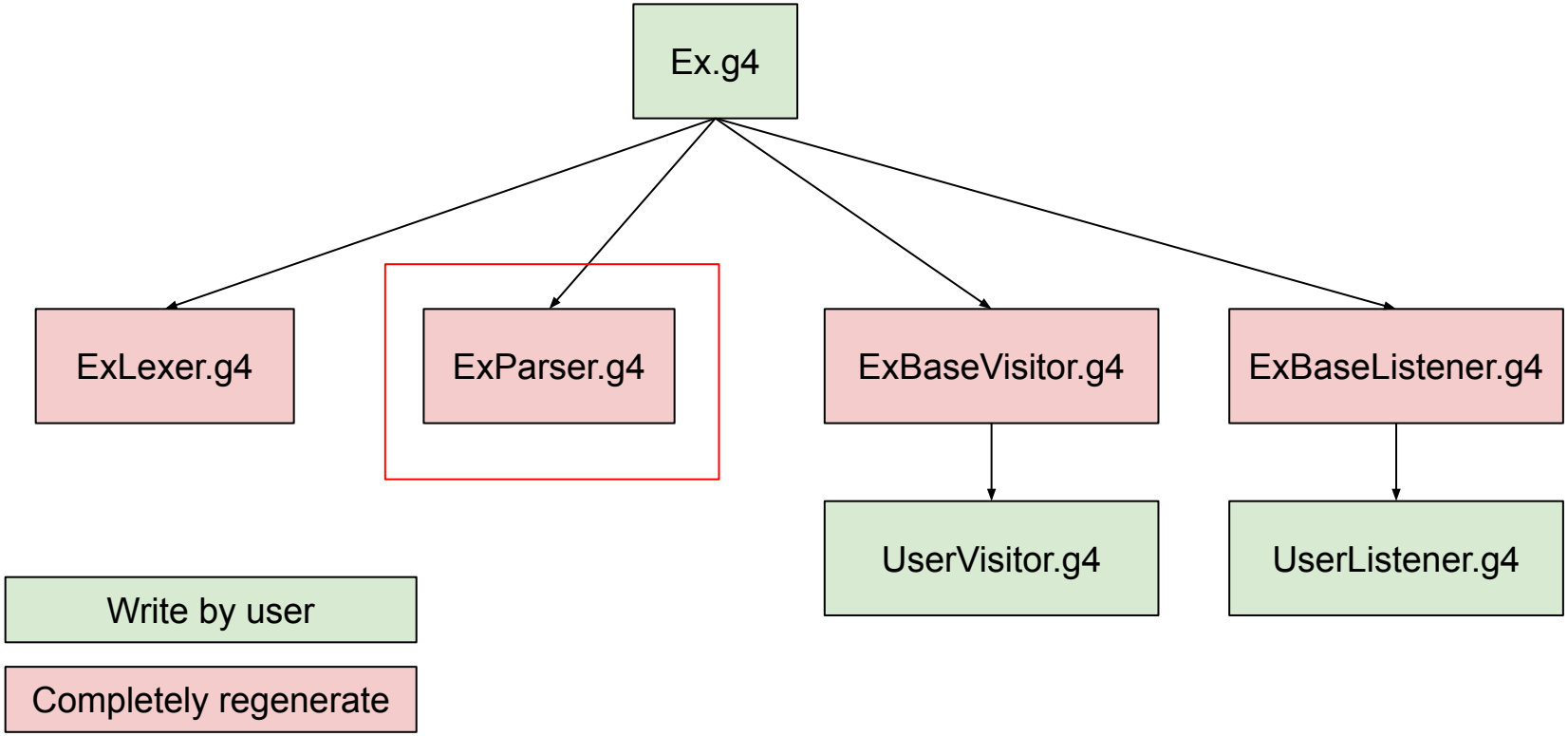
Generation

```
// generates class ExParser
grammar Ex;

//PARSER RULES
id : ID;
program: expr EOF;
expr: expr '*' expr
    | expr '+' expr
    | id
    ;

//LEXER RULES
ID : [A-Za-z]+ ;              // match id with upper, lowercase
WS : [ \t\r\n]+ -> skip ;     // ignore whitespace
```

Generation


Ex.g4
ExLexer.g4
ExParser.g4
ExBaseVisitor.g4
ExBaseListener.g4
UserVisitor.g4
UserListener.g4
Write by user
Completely regenerate

Generation


Ex.g4
ExLexer.g4
ExParser.g4
ExBaseVisitor.g4
ExBaseListener.g4
UserVisitor.g4
UserListener.g4
Write by user
Completely regenerate

Grammar Actions: members

Ex.g4

```
// generates class ExParser
grammar Ex;

@parser::members {
// add members to generated Ex Parser
int col;
public ExParser(TokenStream input, int col)
{
   // custom constructor
   this(input);
   this.col = col;
}
}

...

id : ID;
program: expr EOF;
expr: expr '*' expr
   | expr '+' expr
   | id
   ;
```

ExParser.g4

```
public class ExParser extends Parser {

...

// add members to generated Ex Parser
int col;
public ExParser(TokenStream input, int col) {
   // custom constructor
   this(input);
   this.col = col;
}

...
```

Calculations

Grammar Actions

```
 expr
    : expr '*' expr
    | expr '-' expr
    | INT
    | ID
    ;

program :  assign+ EOF;
assign : ID '=' expr;
```

Grammar Actions: returns

```
expr returns [int v]
    : expr '*' expr
    | expr '-' expr
    | INT
    | ID
    ;
```

```
public static class ExprContext extends ParserRuleContext {
    public int v;
        ...
```

```
program :  assign+ EOF;
assign : ID '=' expr;
```

Grammar Actions: calculation

```
 expr returns [int v]
    : expr '*' expr                    {$v = $l.v * $r.v;}
    | expr '-' expr                    {$v = $l.v - $r.v;}
    | INT
    | ID
    ;

program :  assign+ EOF;
assign : ID '=' expr;
```

Grammar Actions: labels

```
expr returns [int v]
    : l = expr '*' r = expr          {$v = $l.v * $r.v;}
    | l = expr '-' r = expr          {$v = $l.v - $r.v;}
    | INT
    | ID
    ;

program :  assign+ EOF;
assign : ID '=' expr;
```

Grammar Actions: token values

```
expr returns [int v]
    : l = expr '*' r = expr          {$v = $l.v * $r.v;}
    | l = expr '-' r = expr          {$v = $l.v - $r.v;}
    | INT                            {$v = $INT.int; }
    | ID                             {$v =???;}
    ;
```

```
Integer
  .valueOf(((ExprContext)_localctx)
    .INT.getText())
```

```
program :  assign+ EOF;
assign : ID '=' expr;
```

Grammar Actions: members

```
expr returns [int v]
    : l = expr '*' r = expr          {$v = $l.v * $r.v;}
    | l = expr '-' r = expr          {$v = $l.v - $r.v;}
    | INT                            {$v = $INT.int; }
    | ID                             {$v = data.get($ID.text);}
    ;

program :  assign+ EOF;
assign : ID '=' expr {data.put($ID.text, $expr.v);} ;
```

Grammar Actions: members

```
 expr returns [int v]
    : l = expr '*' r = expr          {$v = $l.v * $r.v;}
    | l = expr '-' r = expr          {$v = $l.v - $r.v;}
    | INT                            {$v = $INT.int; }
    | ID                             {$v = data.get($ID.text);}
    ;

program :  assign+ EOF;
assign : ID '=' expr {data.put($ID.text, $expr.v);} ;


@parser::members {
private HashMap<String, Integer> data = new HashMap<>();
}
```

Grammar Actions: header

```
 expr returns [int v]
    : l = expr '*' r = expr          {$v = $l.v * $r.v;}
    | l = expr '-' r = expr          {$v = $l.v - $r.v;}
    | INT                            {$v = $INT.int; }
    | ID                             {$v = data.get($ID.text);}
    ;

program :  assign+ EOF;
assign : ID '=' expr {data.put($ID.text, $expr.v);} ;


@parser::members {
private HashMap<String, Integer> data = new HashMap<>();
}

@header {
package example.gen;
import java.util.HashMap;
}
```

Rule Attributes

Rule Attributes

```
grammar CSV;

file: header row+;

header: row;

row
     :  field (',' field)* '\r'? '\n' ;

field
    : TEXT
    | STRING
    |
    ;

TEXT : ~[,\n\r"]+ ;
STRING : '"' ('""'|~'"')* '"' ;
```

movies.csv

```
"Year", "Score", "Title"
1968,  86, "Greetings"
1970,  17, "Bloody Mama"
1970,  73, "Hi, Mom!"
1971,  40, "Born to Win"
1973,  98, "Mean Streets"
```

Rule Attributes

```
grammar CSV;

file: header (row??)+;

header: row ??;

row [String[] columns]
     :  field (',' field)* '\r'? '\n' ;

field
    : TEXT
    | STRING
    |
    ;

TEXT : ~[,\n\r"]+ ;
STRING : '"' ('""'|~'"')* '"' ;
```

```
public static class RowContext extends ParserRuleContext {
      public RowContext(ParserRuleContext parent,
                        int invokingState,
                        String[] columns) {
            super(parent, invokingState);
            this.columns = columns;
      }
…
}
```

Rule Attributes

```
grammar CSV;

file: header (row[$header.text.split(",")])+;

header: row[null];

row [String[] columns]
     :  field (',' field)* '\r'? '\n'
     {/**map column to headers using columns in Java code**/}
     ;

field
   : TEXT
   | STRING
   |
   ;

TEXT : ~[,\n\r"]+ ;
STRING : '"' ('""'|~'"')* '"' ;
```

java.lang.split

Semantic Predicates

Multiple Language Dialects

java 1.4

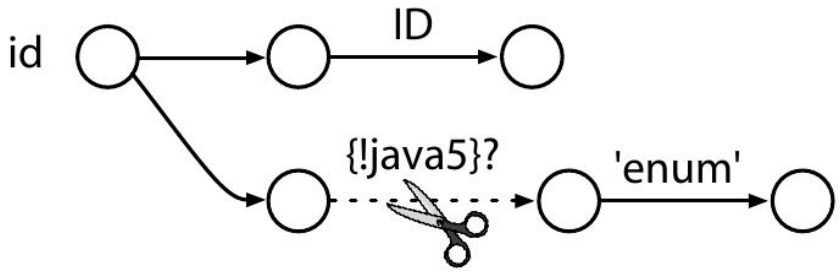
```
String enum = “Enum”;
```

java 1.5

```
enum Temp { HOT, COLD };
```

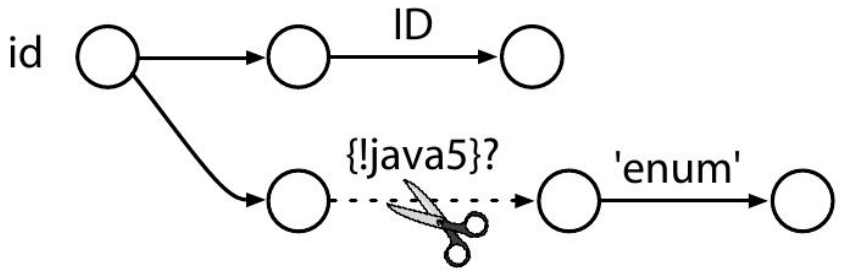
Multiple Language Dialects

```
prog:    ( stat | enumDecl)+;

enumDecl
   :    'enum' name=id '{' id (',' id)* '}';

id  : ID | 'enum';
```



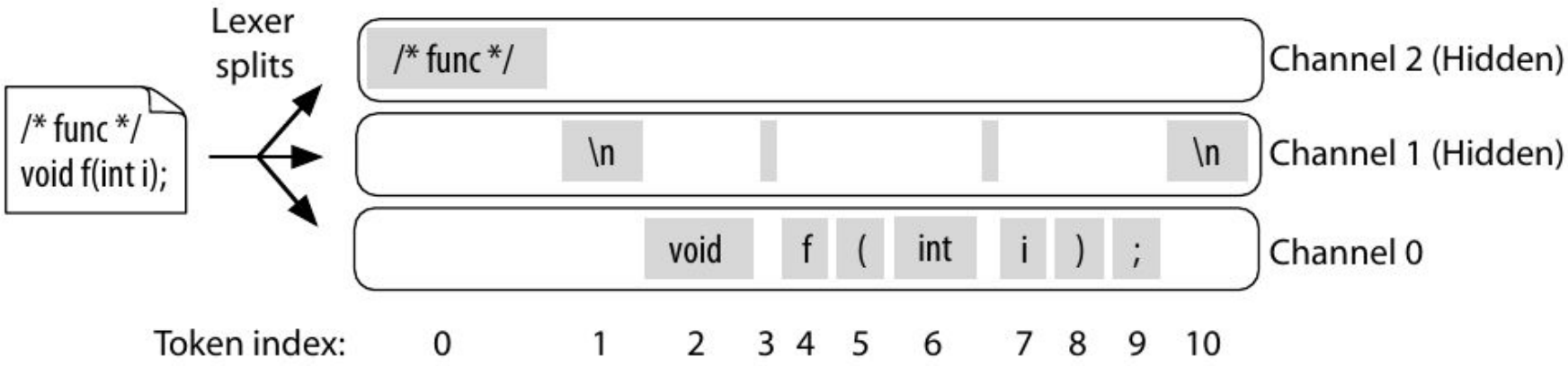
Multiple Language Dialects

```
@parser::members {public static boolean java5;}

prog:    ( stat | enumDecl)+;

enumDecl
   :   {java5}? 'enum' name=id '{' id (',' id)* '}';

id  : ID | {!java5}? 'enum';
```



Lexical modes

Token Channels



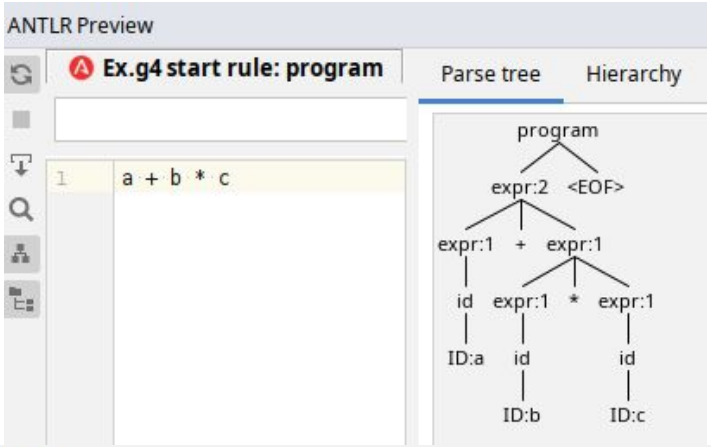

```
WS : [ \t\n\r]+ -> channel(WHITESPACE) ;            // channel(1)
SL_COMMENT: '//' .*? '\n' -> channel(COMMENTS) ;    // channel(2)
```

Island Grammars

```
lexer grammar ModeTagsLexer;
// Default mode rules (the SEA)
OPEN  : '<'     -> mode(ISLAND) ;        // switch to ISLAND mode
TEXT  : ~'<'+ ;                          // clump all text together

mode ISLAND;
CLOSE : '>'     -> mode(DEFAULT_MODE) ; // back to SEA mode
SLASH : '/' ;
ID    : [a-zA-Z]+ ;                      // match/send ID in tag to parser
```

Debug

TestRig



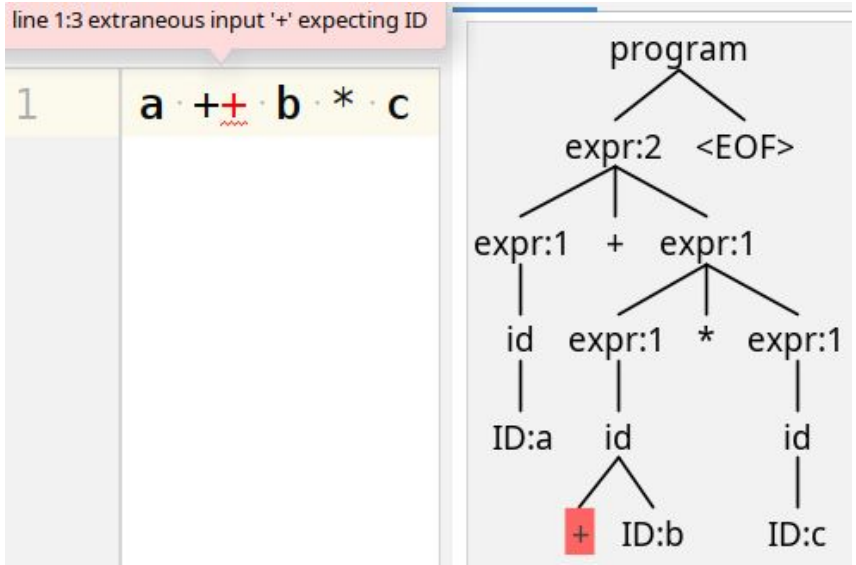
Parse tree | Hierarchy | Profiler

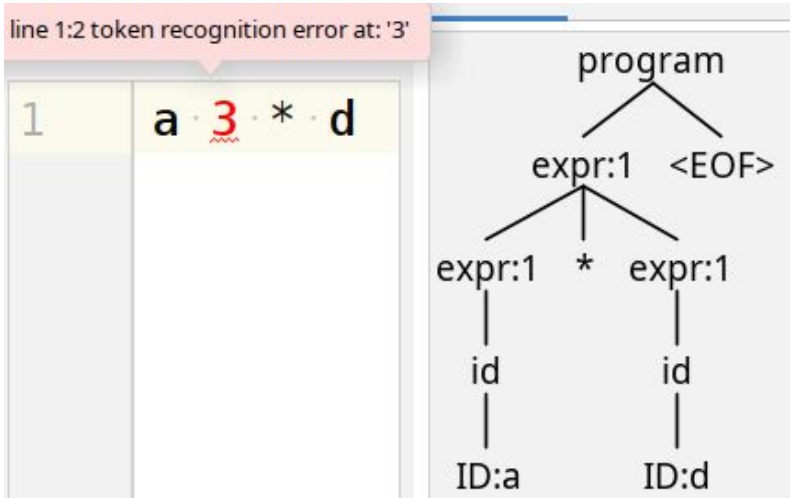
Rule	Invocations	Time	Total k	Max k	Ambiguities	DFA cache...
expr (0)	2	0.104064	2	1	0	2
expr (1)	5	0.249473	5	1	0	5

Input size:	8 char, 1 lines
Number of tokens:	6
Parse time (ms):	1.539
Prediction time (ms):	0.354 = 22.97%
Lookahead burden:	7/6 = 1.17
DFA cache miss rate:	7/7 = 100.00%

☐ Show expert columns

Error Recovery





Error Recovery