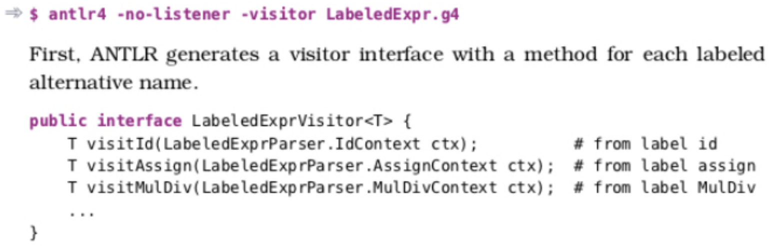
Calculator Implementation with Visitor

- Let’s implement the calculator using the Visitor Pattern



```
⇒ $ antlr4 -no-listener -visitor LabeledExpr.g4
```

First, ANTLR generates a visitor interface with a method for each labeled alternative name.

```
public interface LabeledExprVisitor<T> {
    T visitId(LabeledExprParser.IdContext ctx);           # from label id
    T visitAssign(LabeledExprParser.AssignContext ctx);   # from label assign
    T visitMulDiv(LabeledExprParser.MulDivContext ctx);   # from label MulDiv
    ...
}
```

Calculator Implementation with Visitor

- Subclass `LabeledExprBaseVisitor<T>` with `T` as `Integer`
- Redefine the behaviour of the visit methods
- Create a class with a main that creates a visitor object and visits a parse tree
- See Code...

Translator from Java classes to Java interfaces

- Let’s implement a translator that can parse Java files!
- We are given a Java grammar specification `Java.g4`
- The translator has to transform the code of a Java class into a code for a Java interface containing the same methods without implementation
- Any comment appearing within the method signature must be retained

tour/Java.g4

```
classDeclaration
    :   'class' Identifier typeParameters? ('extends' type)?
        ('implements' typeList)?
        classBody
    ;
```

tour/Java.g4

```
methodDeclaration
    :   type Identifier formalParameters ('[' ']')* methodDeclarationRest
    |   'void' Identifier formalParameters methodDeclarationRest
    ;
```

Translator from Java classes to Java interfaces

tour/Demo.java

```
import java.util.List;
import java.util.Map;
public class Demo {
        void f(int x, String y) { }
        int[ ] g(/*no args*/) { return null; }
        List<Map<String, Integer>>[] h() { return null; }
}
```

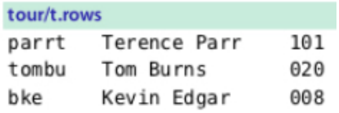
must produce (see code):

tour/IDemo.java

```
interface IDemo {
        void f(int x, String y);
        int[ ] g(/*no args*/);
        List<Map<String, Integer>>[] h();
}
```

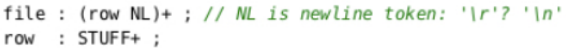
Implementing an SDT in ANTLR4

- Let’s implement a translator that parses a csv text file with tab as separator
- We want to select the data values of a particular column


tour/t.rows

```
parrt	Terence Parr	101
tombu	Tom Burns	020
bke	Kevin Edgar	008
```

Base grammar:



```
file : (row NL)+ ; // NL is newline token: '\r'? '\n'
row  : STUFF+ ;
```

Implementing an SDT in ANTLR4

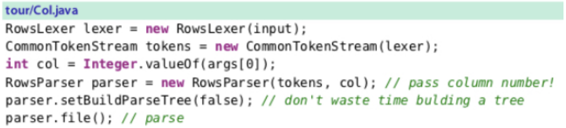
- Enriched grammar with code

tour/Rows.g4

```
grammar Rows;

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

file: (row NL)+ ;

row
locals [int i=0]
    : (   STUFF
          {
          $i++;
          if ( $i == col ) System.out.println($STUFF.text);
          }
      )+
    ;

TAB  :  '\t' -> skip ;   // match but don't pass to the parser
NL   :  '\r'? '\n' ;     // match and pass to the parser
STUFF:  ~[\t\r\n]+ ;     // match any chars except tab, newline
```

Implementing an SDT in ANTLR4

- Running the parser (see code)



tour/Col.java

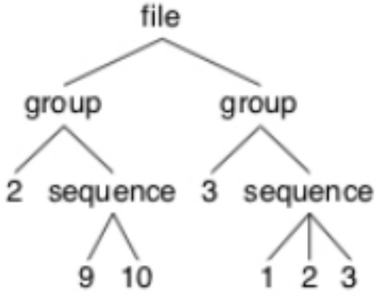
```
RowsLexer lexer = new RowsLexer(input);
CommonTokenStream tokens = new CommonTokenStream(lexer);
int col = Integer.valueOf(args[0]);
RowsParser parser = new RowsParser(tokens, col); // pass column number!
parser.setBuildParseTree(false); // don't waste time bulding a tree
parser.file(); // parse
```

Adaptive Parsing in ANTLR4

- With ANTLR4 it is possible to modify the behaviour of the parser at runtime, depending on the parsed input
- Technically this is realised through semantic predicates, which are boolean expressions enclosed in curly braces followed by a question mark, e.g. `{$i <= n}?`
- Suppose we want to parse a sequence of integers such that some of them tells how to group them: `2 9 10 3 1 2 3`
- the first `2` means that we want to group the following two numbers
- then the following `3` means that we want to group the following three numbers

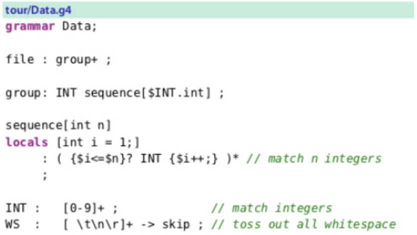
Adaptive Parsing in ANTLR4

- We want that the parsing of `2 9 10 3 1 2 3` produces the following parse tree



Adaptive Parsing in ANTLR4

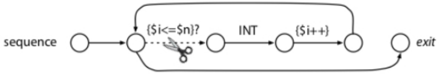
- The grammar to produce this result is the following



tour/Data.g4

```
grammar Data;

file : group+ ;

group: INT sequence[$INT.int] ;

sequence[int n]
locals [int i = 1;]
     : ( {$i<=$n}? INT {$i++;} )* // match n integers
     ;

INT :   [0-9]+ ;             // match integers
WS  :   [ \t\n\r]+ -> skip ; // toss out all whitespace
```

- Visual representation of the effect of the semantic predicate



- See produced code in the parser class