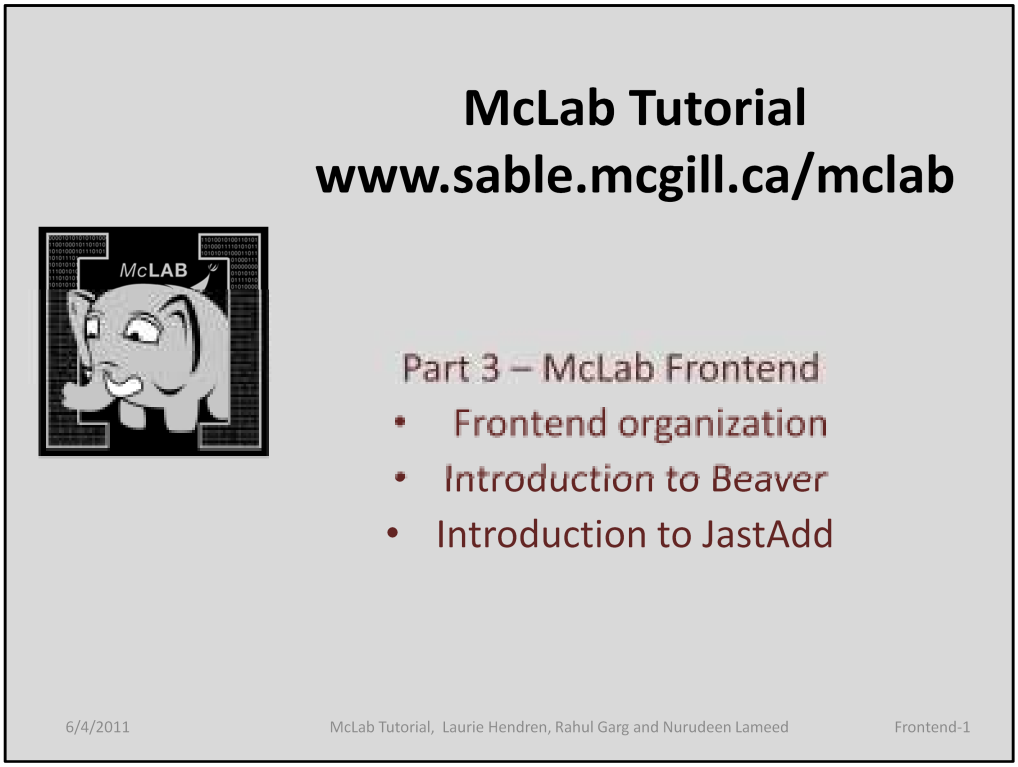
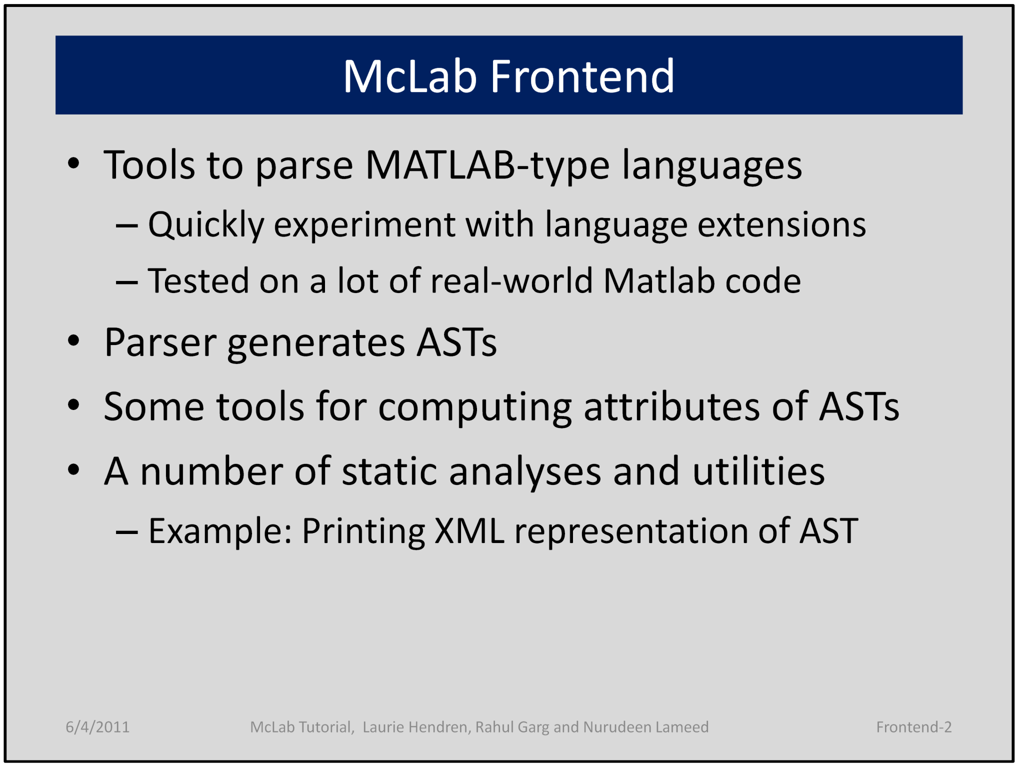


McLab Tutorial
www.sable.mcgill.ca/mclab

Part 3 – McLab Frontend

- Frontend organization
- Introduction to Beaver
- Introduction to JastAdd

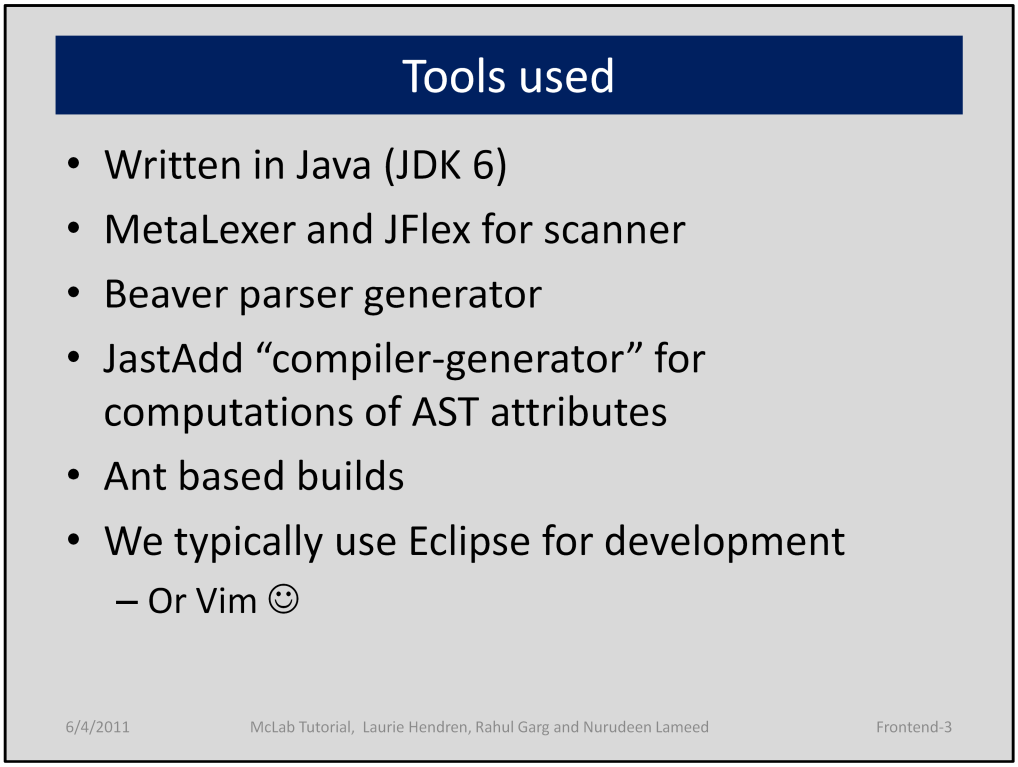
6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-1



McLab Frontend

- Tools to parse MATLAB-type languages
  - Quickly experiment with language extensions
  - Tested on a lot of real-world Matlab code
- Parser generates ASTs
- Some tools for computing attributes of ASTs
- A number of static analyses and utilities
  - Example: Printing XML representation of AST

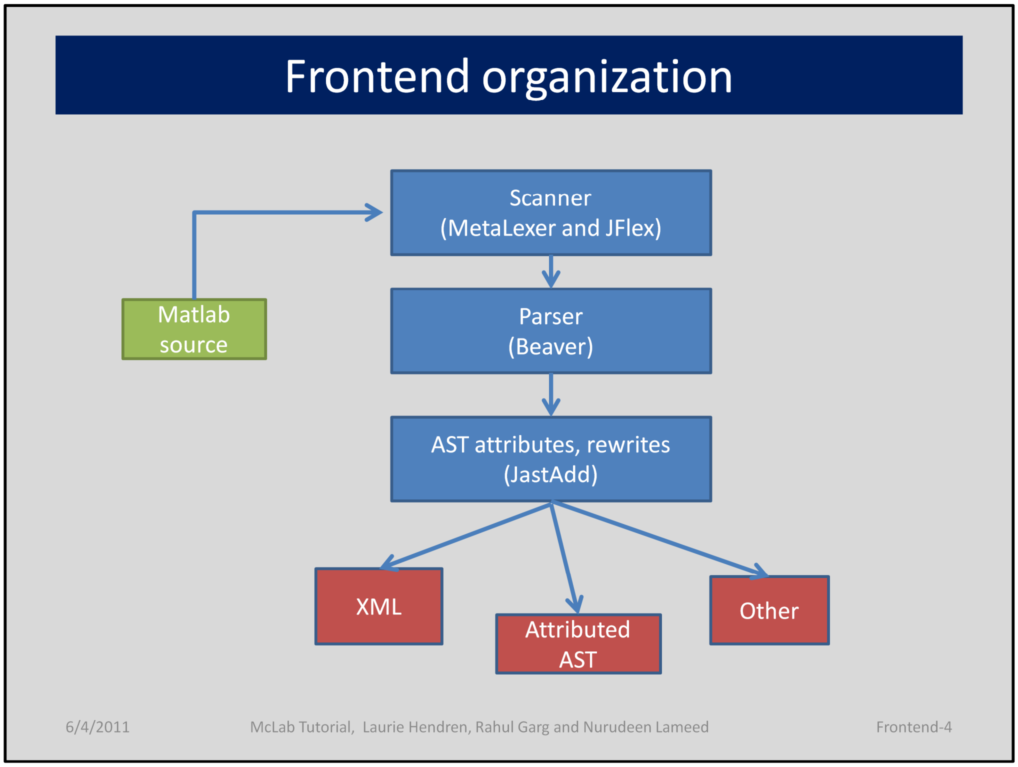
6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-2



Tools used

- Written in Java (JDK 6)
- MetaLexer and JFlex for scanner
- Beaver parser generator
- JastAdd “compiler-generator” for computations of AST attributes
- Ant based builds
- We typically use Eclipse for development
  - Or Vim ☺

Look! Notes!



Frontend organization

Matlab source

Scanner (MetaLexer and JFlex)

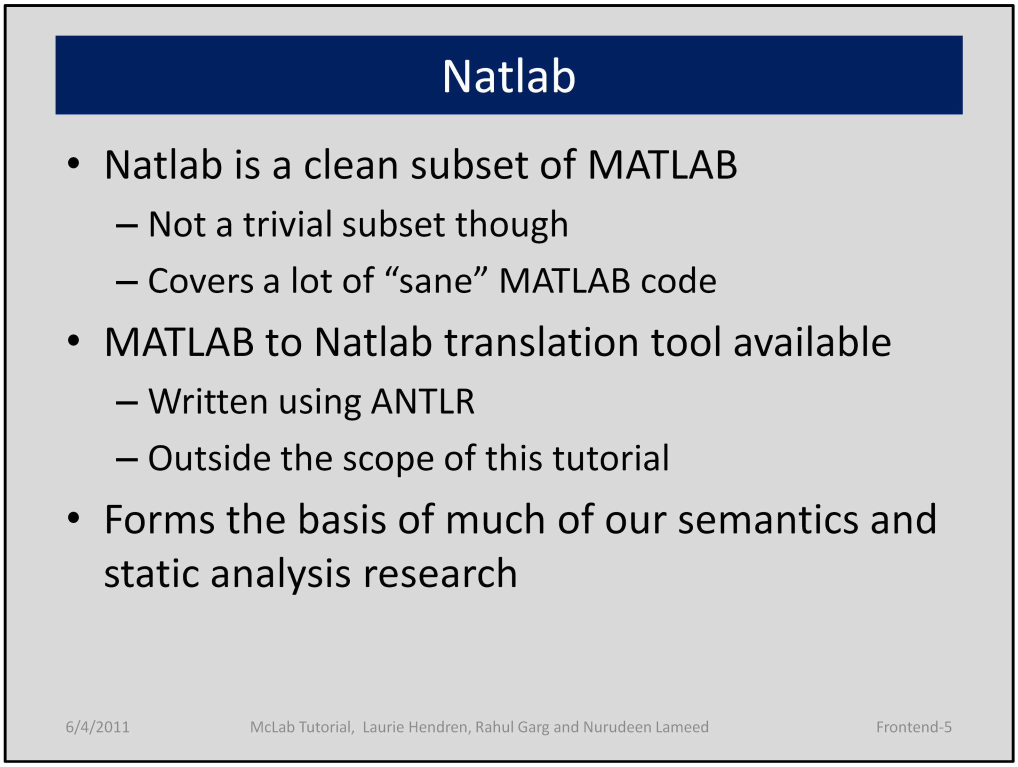
Parser (Beaver)

AST attributes, rewrites (JastAdd)

XML

Attributed AST

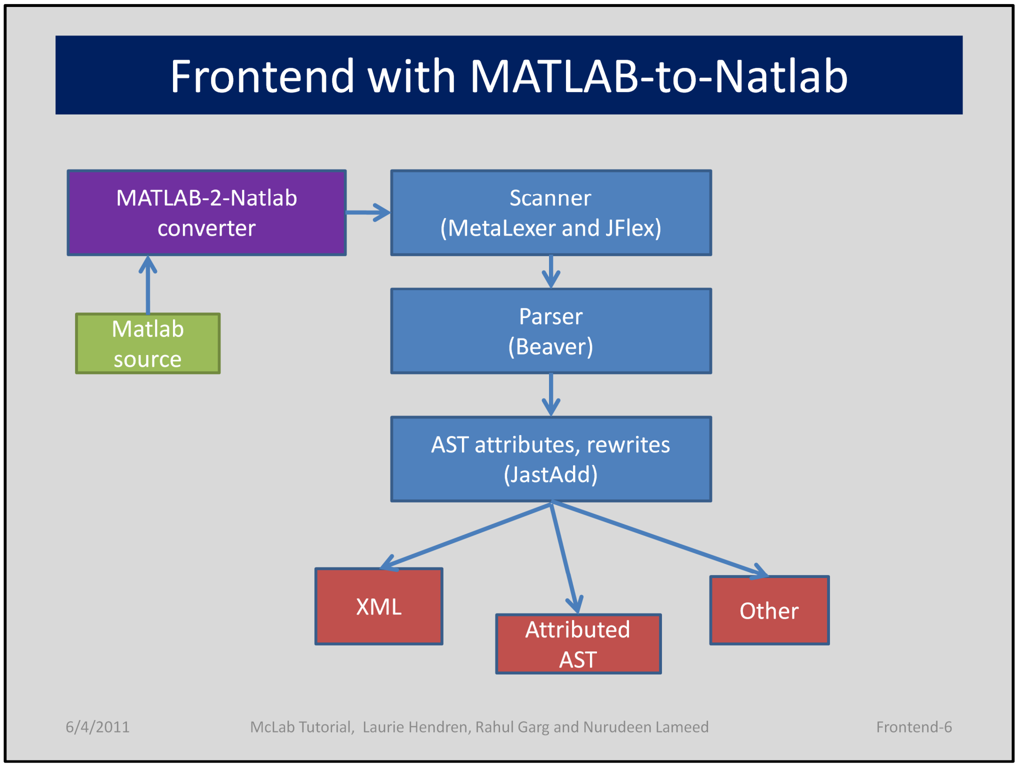
Other



Natlab

- Natlab is a clean subset of MATLAB
  - Not a trivial subset though
  - Covers a lot of “sane” MATLAB code
- MATLAB to Natlab translation tool available
  - Written using ANTLR
  - Outside the scope of this tutorial
- Forms the basis of much of our semantics and static analysis research

Derivatives such as AspectMatlab use the work done in Natlab.



Frontend with MATLAB-to-Natlab

MATLAB-2-Natlab converter

Matlab source

Scanner (MetaLexer and JFlex)

Parser (Beaver)

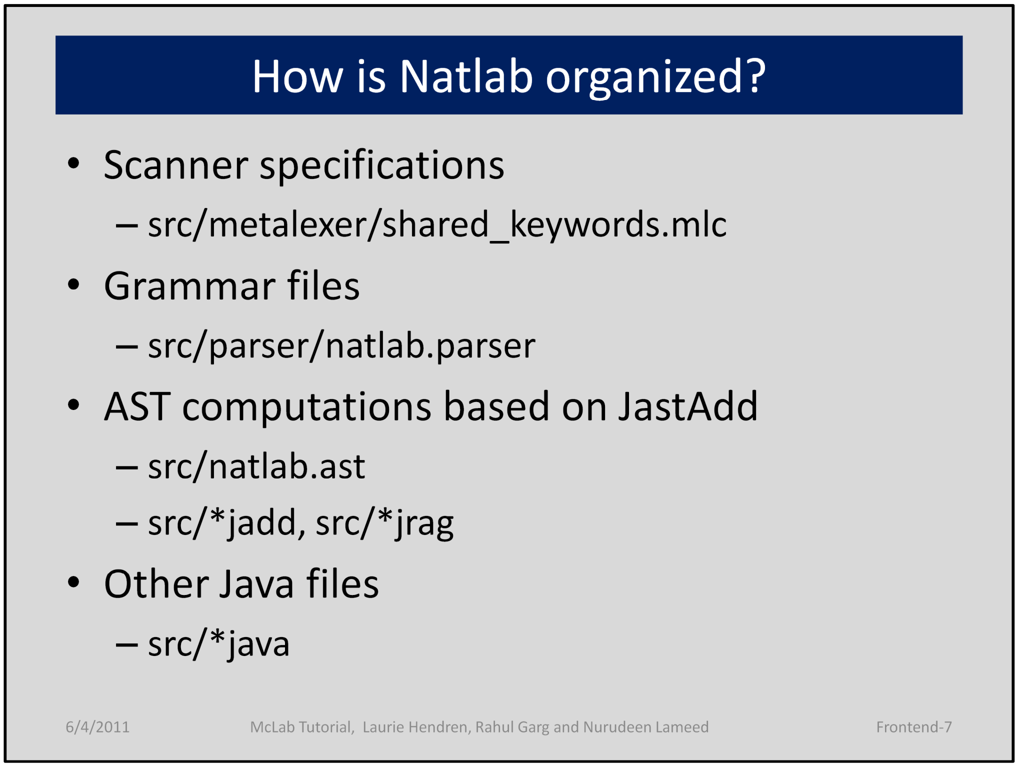
AST attributes, rewrites (JastAdd)

XML

Attributed AST

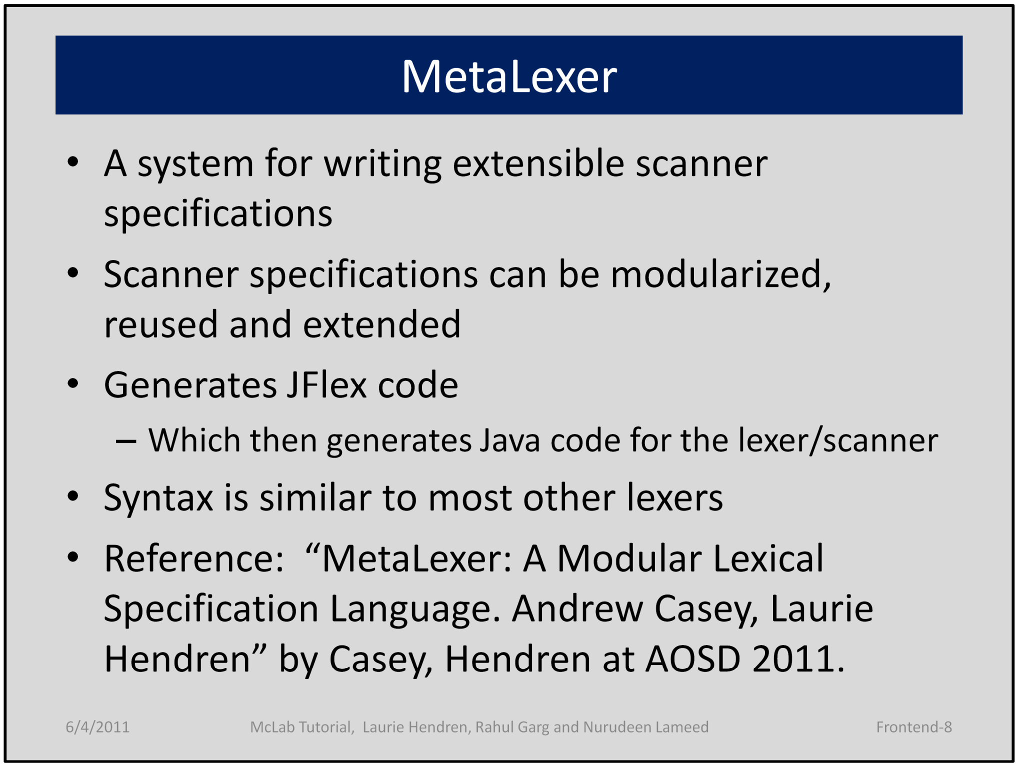
Other

6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-6



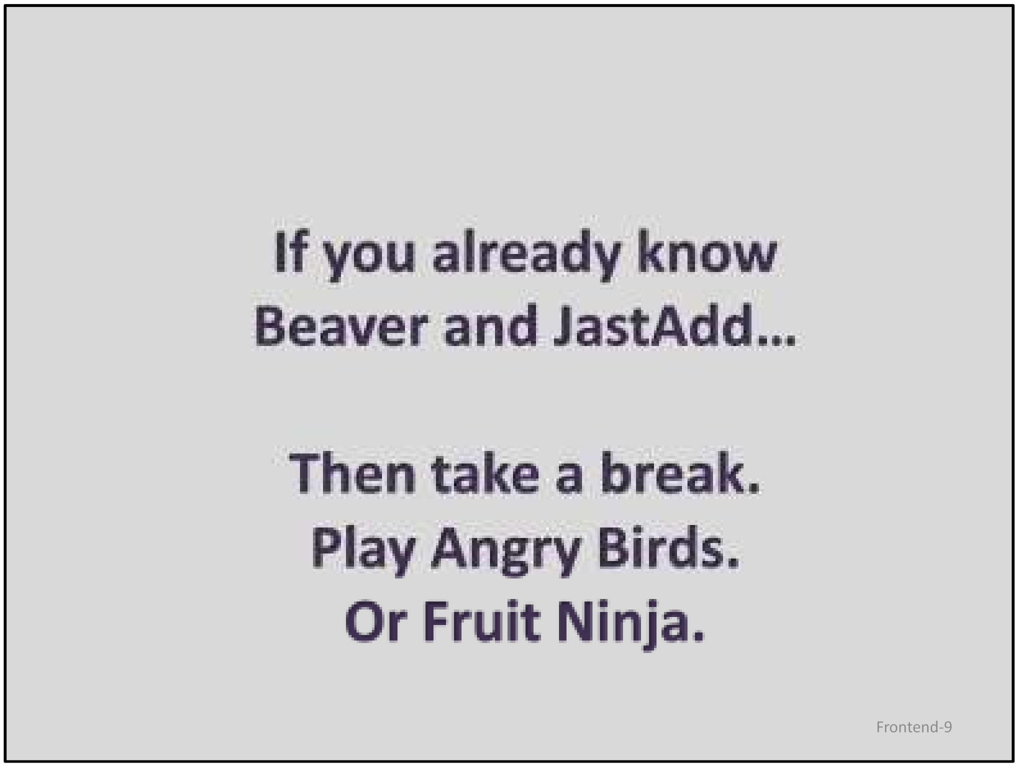
How is Natlab organized?

- Scanner specifications
  - src/metalexer/shared_keywords.mlc
- Grammar files
  - src/parser/natlab.parser
- AST computations based on JastAdd
  - src/natlab.ast
  - src/*jadd, src/*jrag
- Other Java files
  - src/*java



MetaLexer

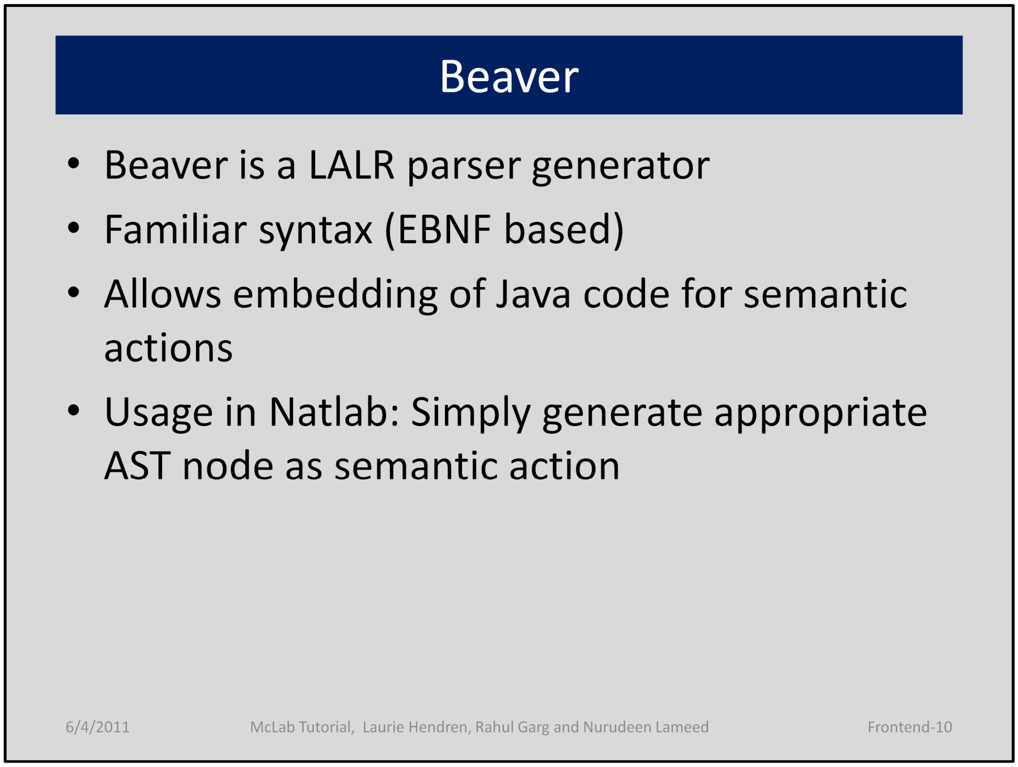
- A system for writing extensible scanner specifications
- Scanner specifications can be modularized, reused and extended
- Generates JFlex code
  - Which then generates Java code for the lexer/scanner
- Syntax is similar to most other lexers
- Reference: “MetaLexer: A Modular Lexical Specification Language. Andrew Casey, Laurie Hendren” by Casey, Hendren at AOSD 2011.



If you already know
Beaver and JastAdd…

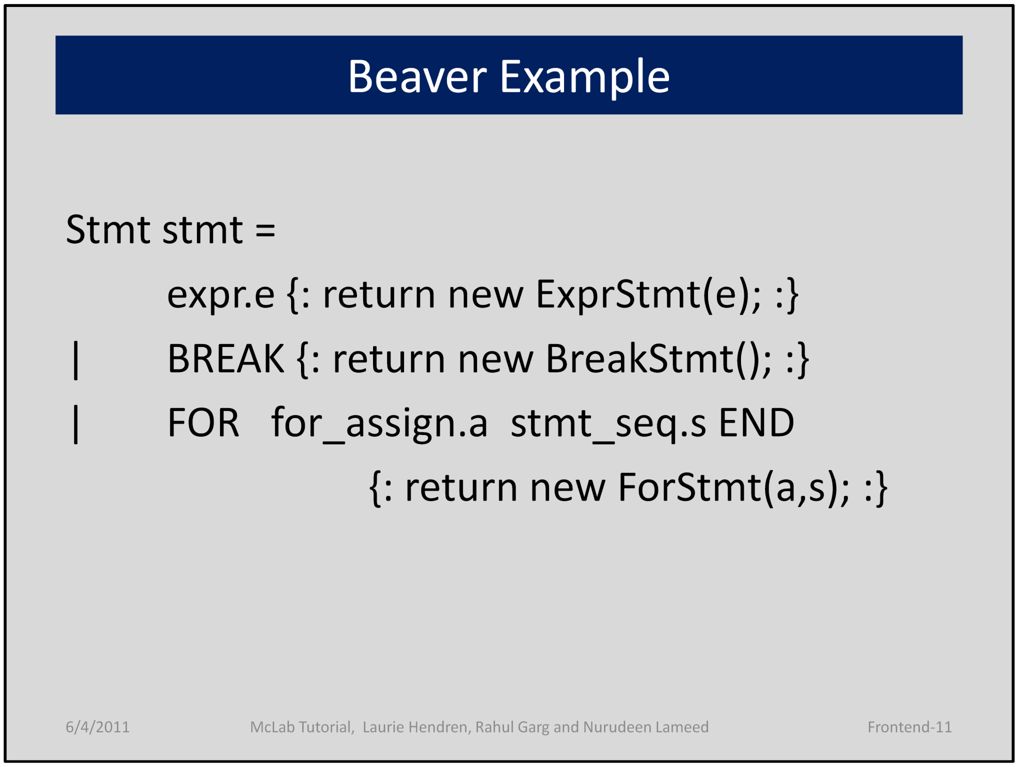
Then take a break.
Play Angry Birds.
Or Fruit Ninja.

Frontend-9



Beaver

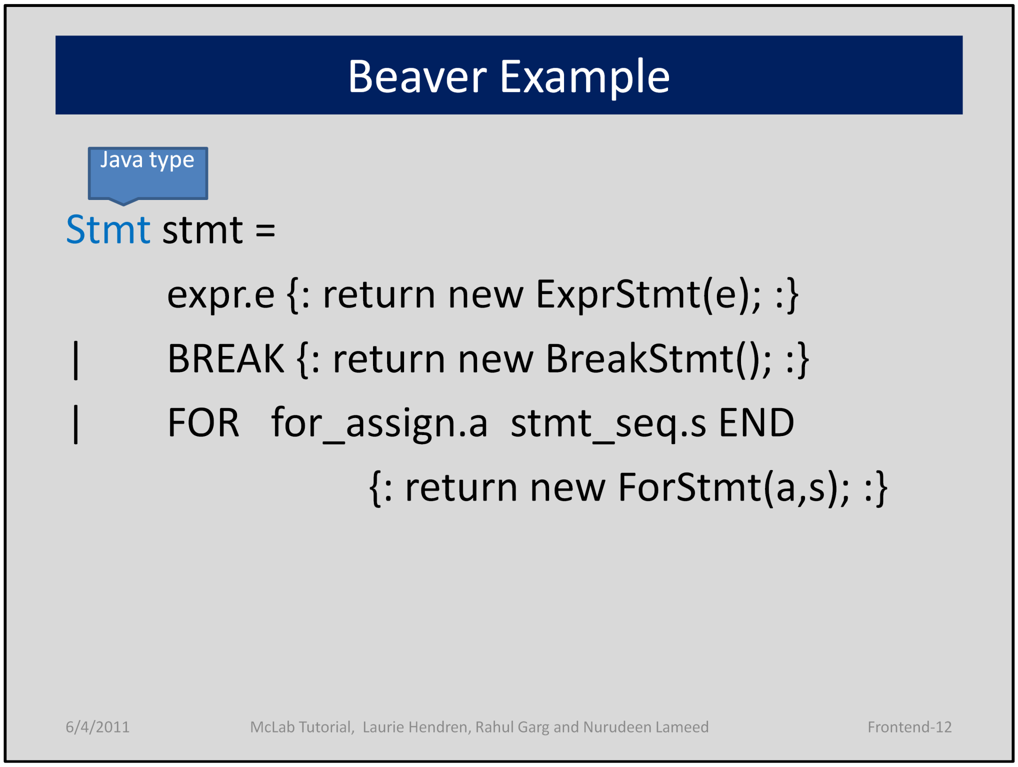
- Beaver is a LALR parser generator
- Familiar syntax (EBNF based)
- Allows embedding of Java code for semantic actions
- Usage in Natlab: Simply generate appropriate AST node as semantic action



Beaver Example

```
Stmt stmt =
        expr.e {: return new ExprStmt(e); :}
|       BREAK {: return new BreakStmt(); :}
|       FOR   for_assign.a  stmt_seq.s END
                        {: return new ForStmt(a,s); :}
```

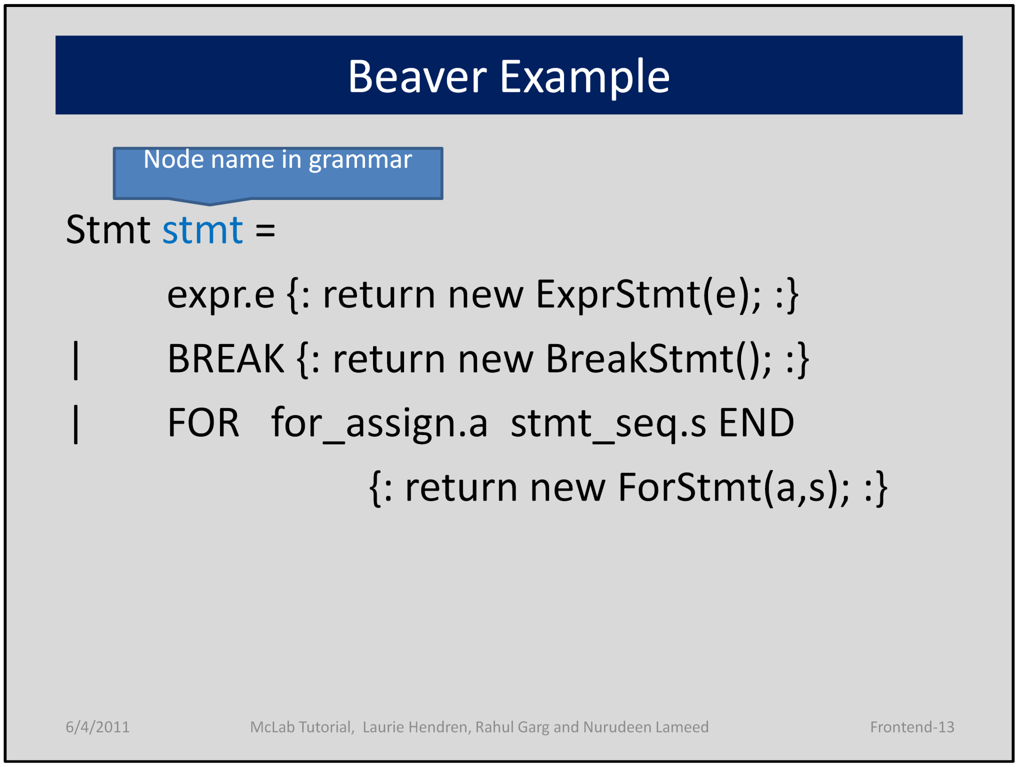
Example is a simplified grammar



Beaver Example

```
Stmt stmt =
        expr.e {: return new ExprStmt(e); :}
|       BREAK {: return new BreakStmt(); :}
|       FOR   for_assign.a  stmt_seq.s END
                         {: return new ForStmt(a,s); :}
```

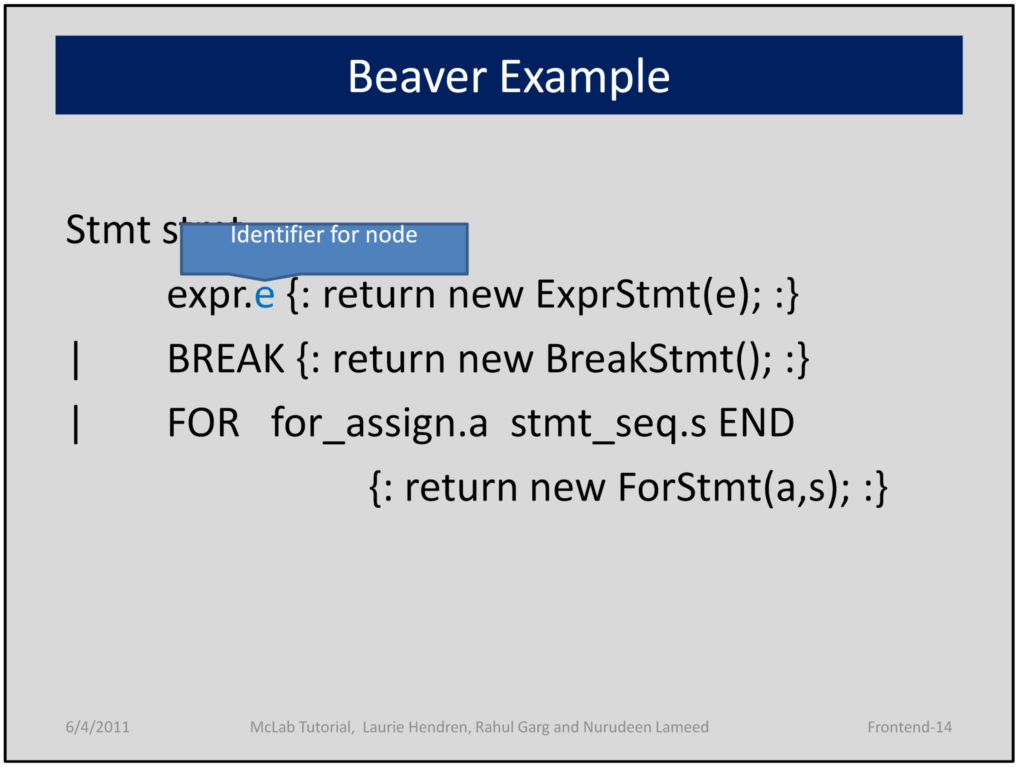
The Java types must be declared/defined/imported by the programmer.



Beaver Example

Node name in grammar

```
Stmt stmt =
        expr.e {: return new ExprStmt(e); :}
|       BREAK {: return new BreakStmt(); :}
|       FOR   for_assign.a  stmt_seq.s END
                        {: return new ForStmt(a,s); :}
```



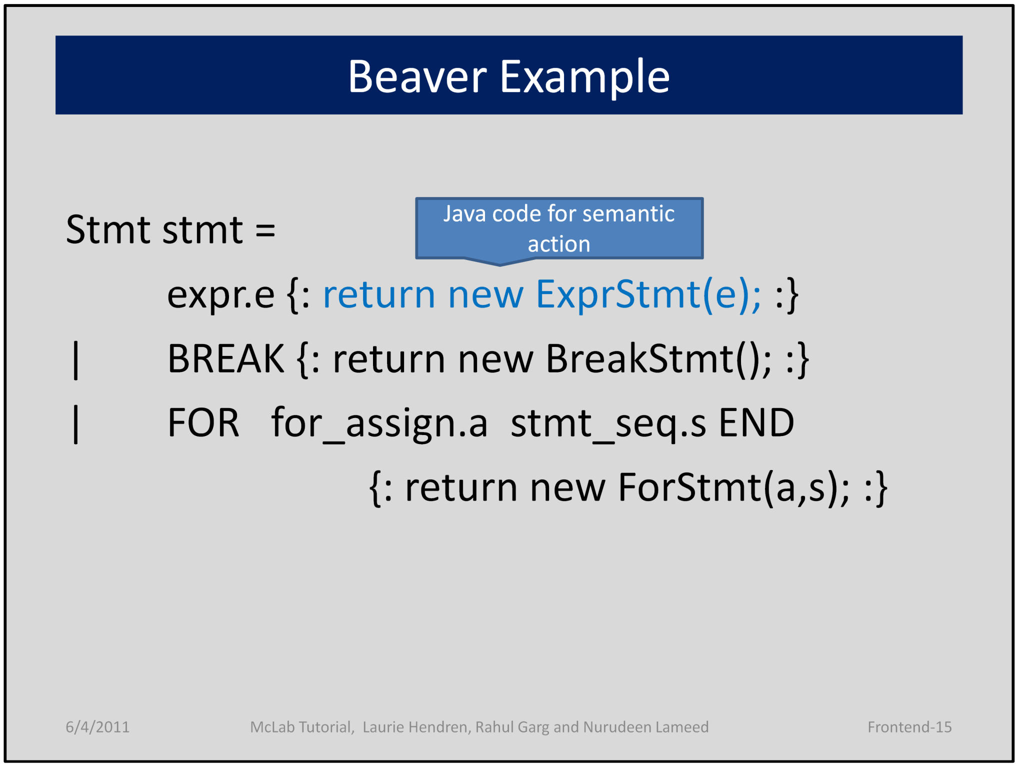
Beaver Example

```
Stmt s
        expr.e {: return new ExprStmt(e); :}
|       BREAK {: return new BreakStmt(); :}
|       FOR   for_assign.a  stmt_seq.s END
                         {: return new ForStmt(a,s); :}
```

Identifier for node

6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-14

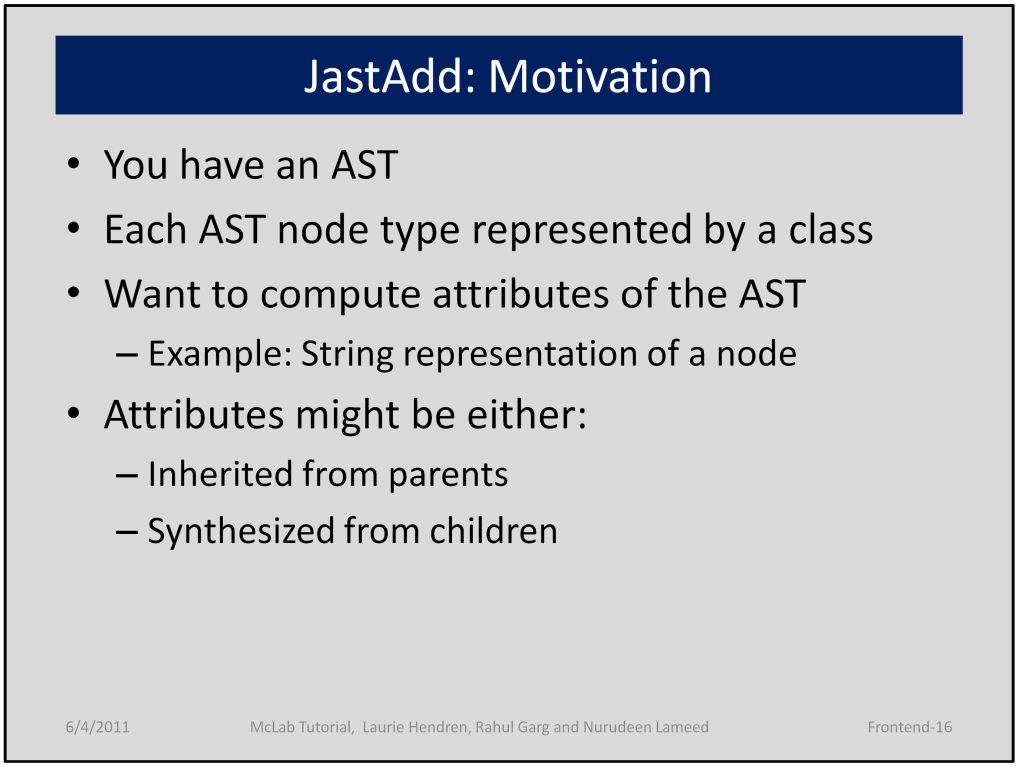
The name given to a node can then be used inside the semantic action.



Beaver Example

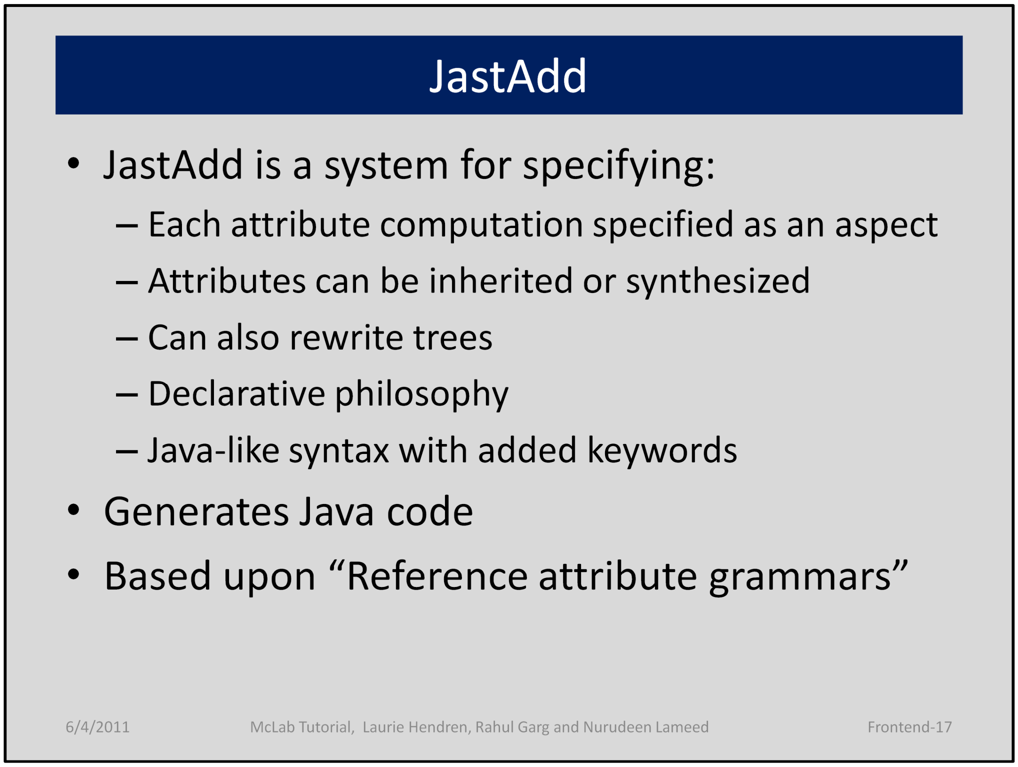
Java code for semantic action

```
Stmt stmt =
        expr.e {: return new ExprStmt(e); :}
|       BREAK {: return new BreakStmt(); :}
|       FOR   for_assign.a  stmt_seq.s END
                        {: return new ForStmt(a,s); :}
```



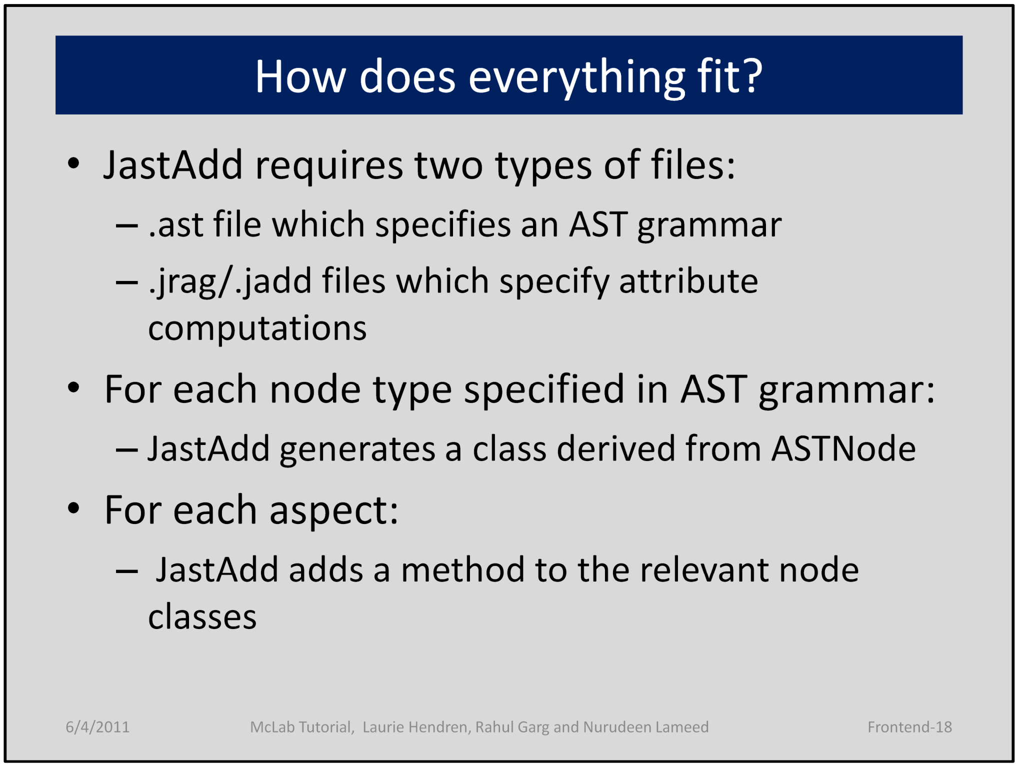
JastAdd: Motivation

- You have an AST
- Each AST node type represented by a class
- Want to compute attributes of the AST
  - Example: String representation of a node
- Attributes might be either:
  - Inherited from parents
  - Synthesized from children



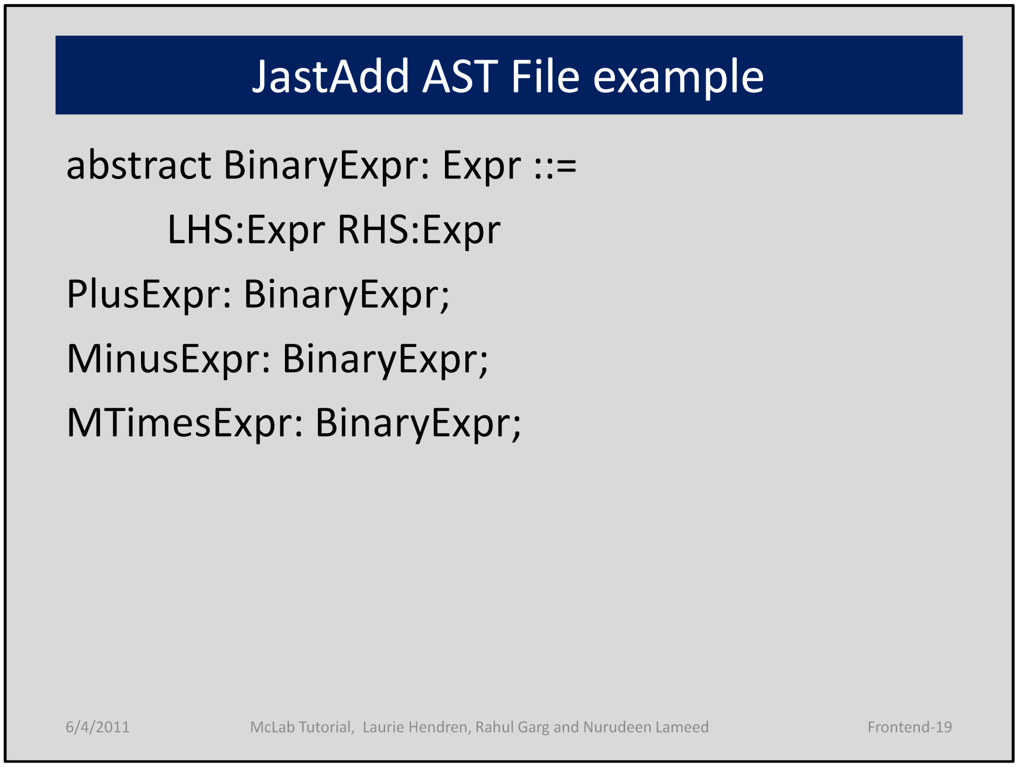
JastAdd

- JastAdd is a system for specifying:
  - Each attribute computation specified as an aspect
  - Attributes can be inherited or synthesized
  - Can also rewrite trees
  - Declarative philosophy
  - Java-like syntax with added keywords
- Generates Java code
- Based upon “Reference attribute grammars”



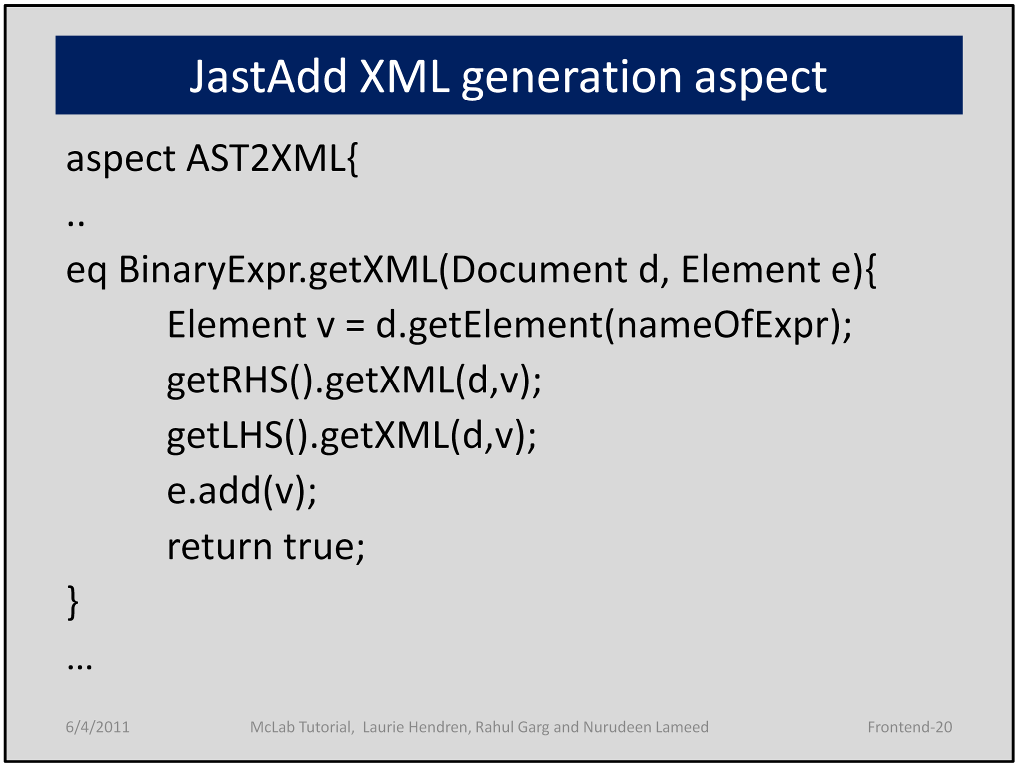
How does everything fit?

- JastAdd requires two types of files:
  - .ast file which specifies an AST grammar
  - .jrag/.jadd files which specify attribute computations
- For each node type specified in AST grammar:
  - JastAdd generates a class derived from ASTNode
- For each aspect:
  - JastAdd adds a method to the relevant node classes



JastAdd AST File example

```
abstract BinaryExpr: Expr ::=
        LHS:Expr RHS:Expr
PlusExpr: BinaryExpr;
MinusExpr: BinaryExpr;
MTimesExpr: BinaryExpr;
```

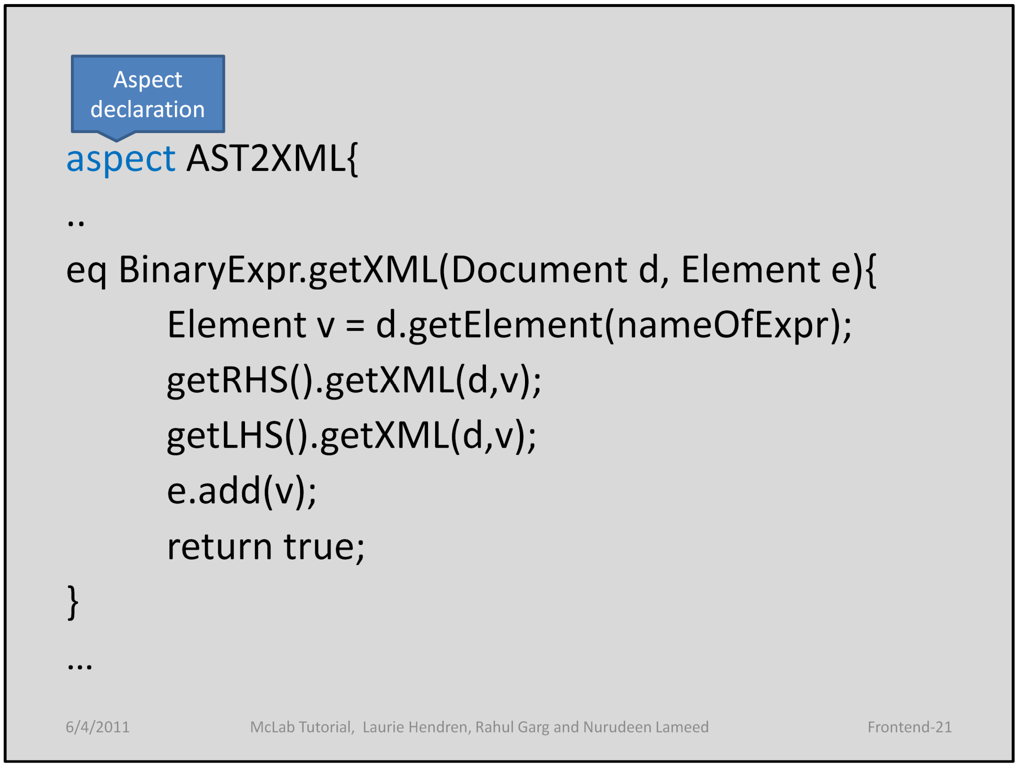


JastAdd XML generation aspect

```
aspect AST2XML{
..
eq BinaryExpr.getXML(Document d, Element e){
        Element v = d.getElement(nameOfExpr);
        getRHS().getXML(d,v);
        getLHS().getXML(d,v);
        e.add(v);
        return true;
}
…
```

6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-20

Code has been simplified to suit the purposes of the tutorial. Actual code will do a little more bookkeeping of line numbers etc.

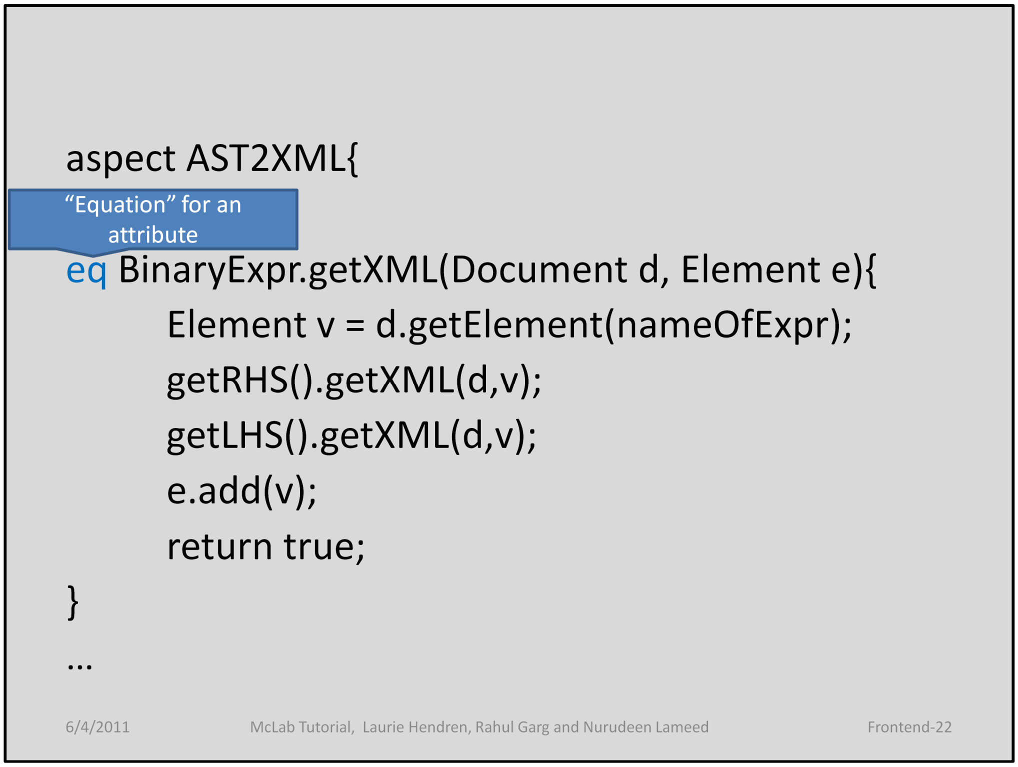


Aspect declaration

```
aspect AST2XML{
..
eq BinaryExpr.getXML(Document d, Element e){
        Element v = d.getElement(nameOfExpr);
        getRHS().getXML(d,v);
        getLHS().getXML(d,v);
        e.add(v);
        return true;
}
...
```

6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-21

Code has been simplified to suit the purposes of the tutorial. Actual code will do a little more bookkeeping of line numbers etc.



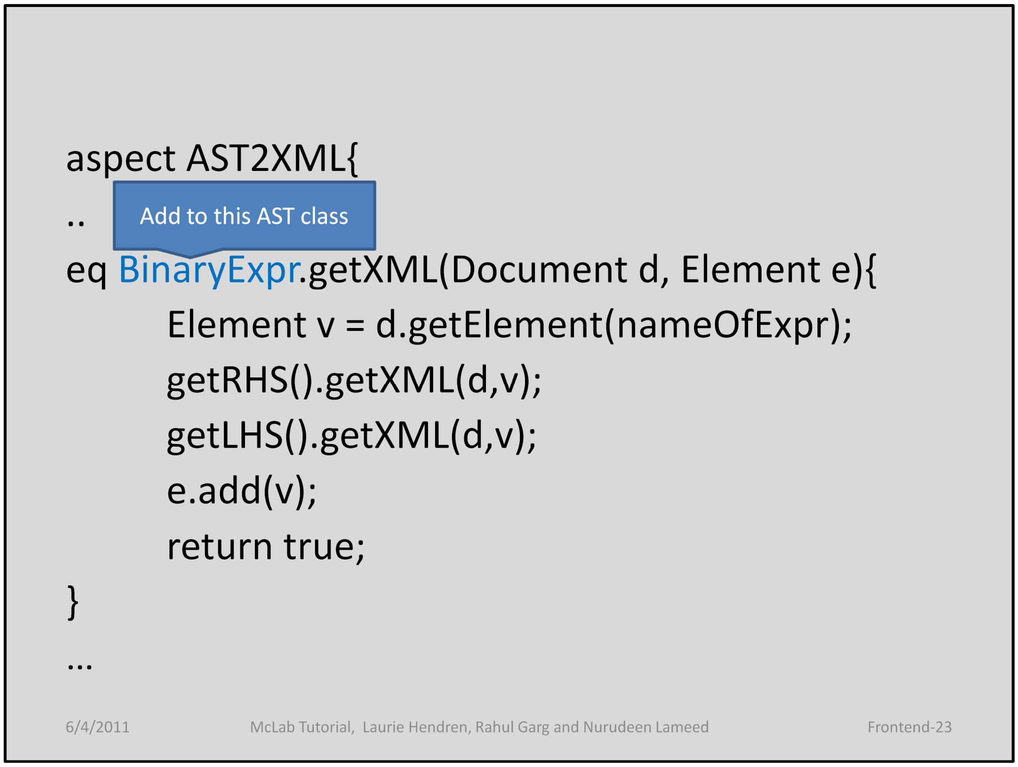
“Equation” for an attribute

```
aspect AST2XML{

eq BinaryExpr.getXML(Document d, Element e){
        Element v = d.getElement(nameOfExpr);
        getRHS().getXML(d,v);
        getLHS().getXML(d,v);
        e.add(v);
        return true;
}
…
```

6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-22

Code has been simplified to suit the purposes of the tutorial. Actual code will do a little more bookkeeping of line numbers etc.

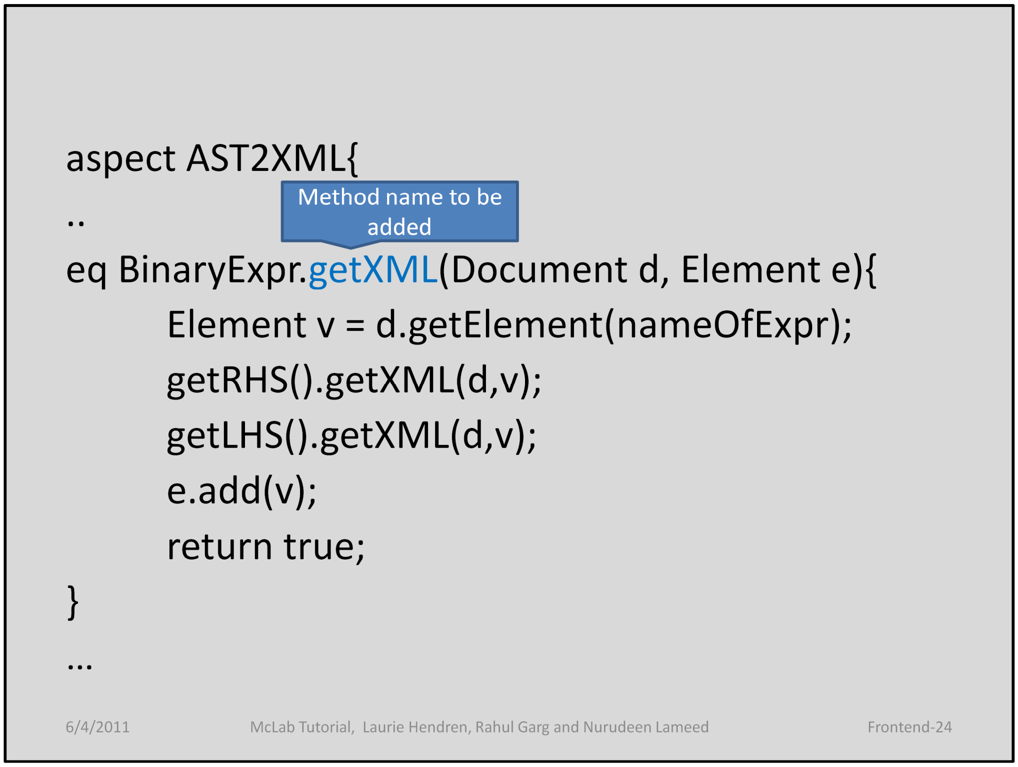


Add to this AST class

```
aspect AST2XML{
..
eq BinaryExpr.getXML(Document d, Element e){
        Element v = d.getElement(nameOfExpr);
        getRHS().getXML(d,v);
        getLHS().getXML(d,v);
        e.add(v);
        return true;
}
...
```

6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-23

Code has been simplified to suit the purposes of the tutorial. Actual code will do a little more bookkeeping of line numbers etc.

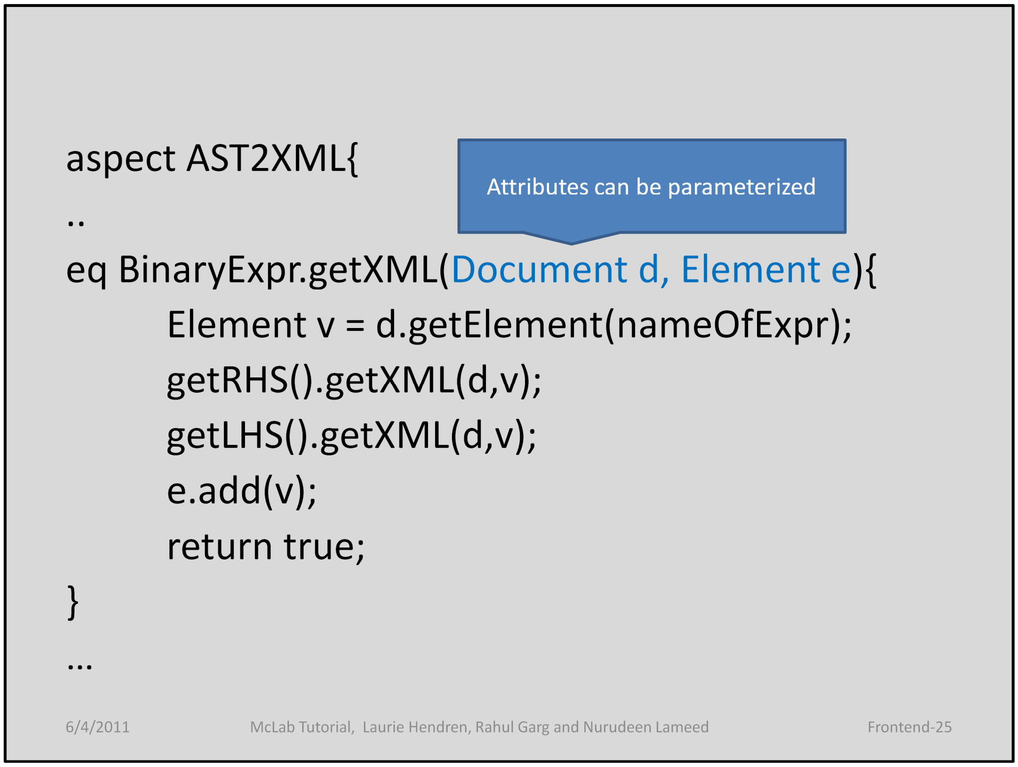


Method name to be added

```
aspect AST2XML{
..
eq BinaryExpr.getXML(Document d, Element e){
        Element v = d.getElement(nameOfExpr);
        getRHS().getXML(d,v);
        getLHS().getXML(d,v);
        e.add(v);
        return true;
}
...
```

6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-24

Code has been simplified to suit the purposes of the tutorial. Actual code will do a little more bookkeeping of line numbers etc.

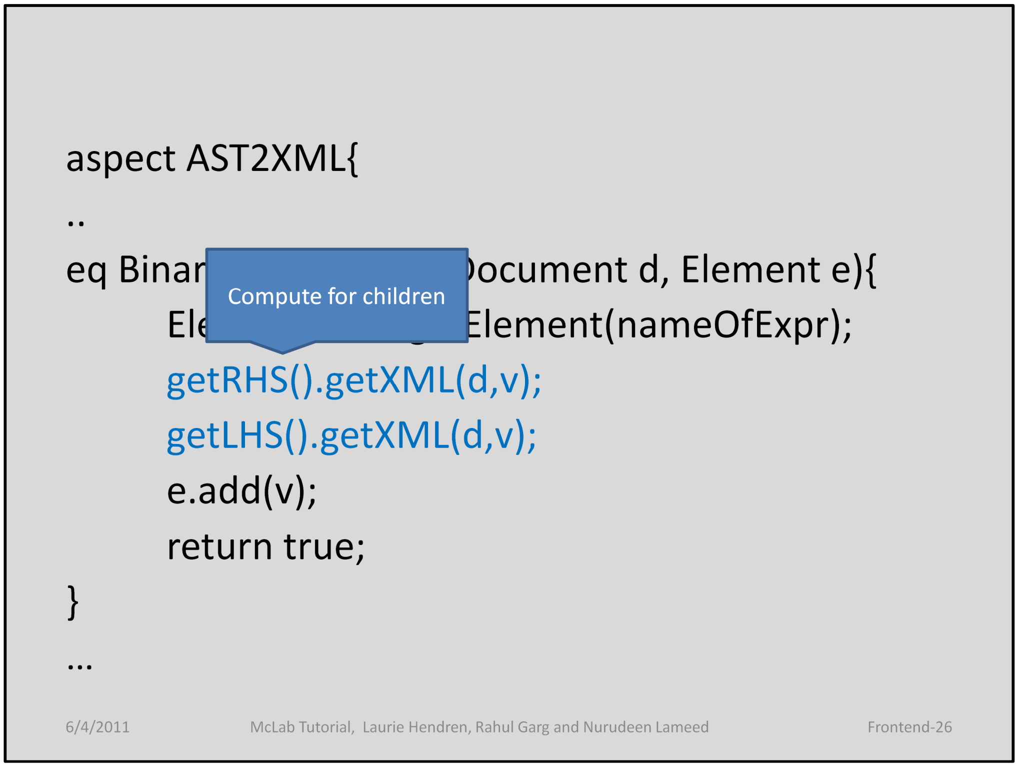


```
aspect AST2XML{
..
eq BinaryExpr.getXML(Document d, Element e){
        Element v = d.getElement(nameOfExpr);
        getRHS().getXML(d,v);
        getLHS().getXML(d,v);
        e.add(v);
        return true;
}
...
```

Attributes can be parameterized

6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-25

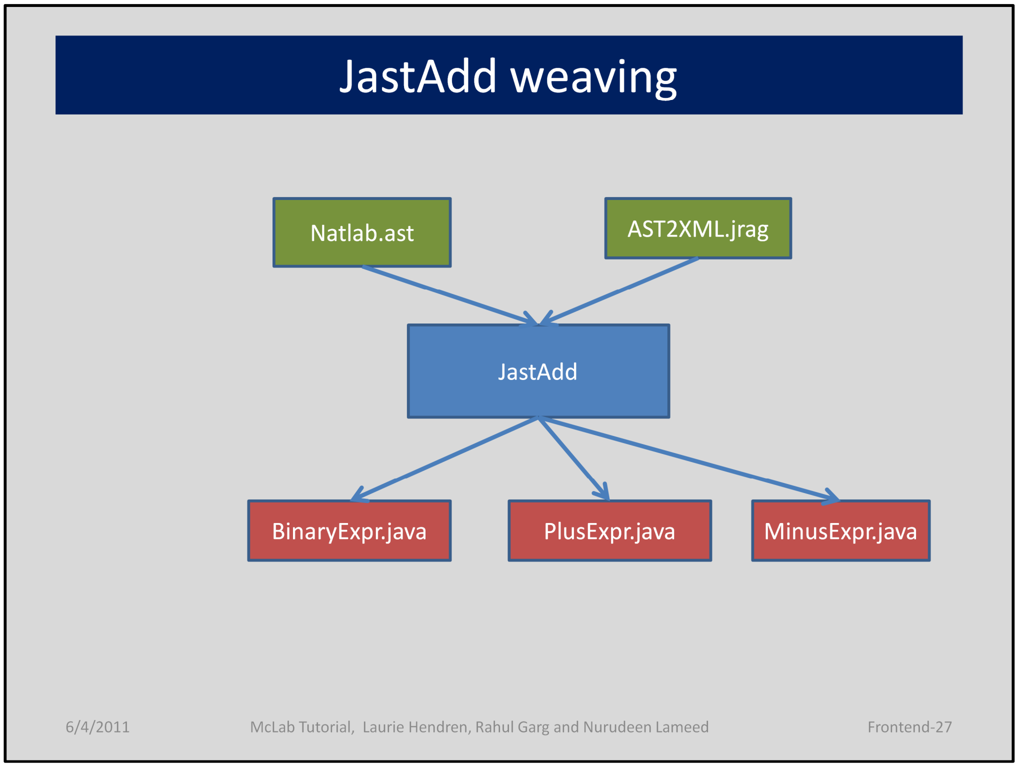
Code has been simplified to suit the purposes of the tutorial. Actual code will do a little more bookkeeping of line numbers etc.



```
aspect AST2XML{
..
eq Binar                    ocument d, Element e){
        Ele                Element(nameOfExpr);
        getRHS().getXML(d,v);
        getLHS().getXML(d,v);
        e.add(v);
        return true;
}
…
```

Compute for children

Code has been simplified to suit the purposes of the tutorial. Actual code will do a little more bookkeeping of line numbers etc.



JastAdd weaving

Natlab.ast

AST2XML.jrag

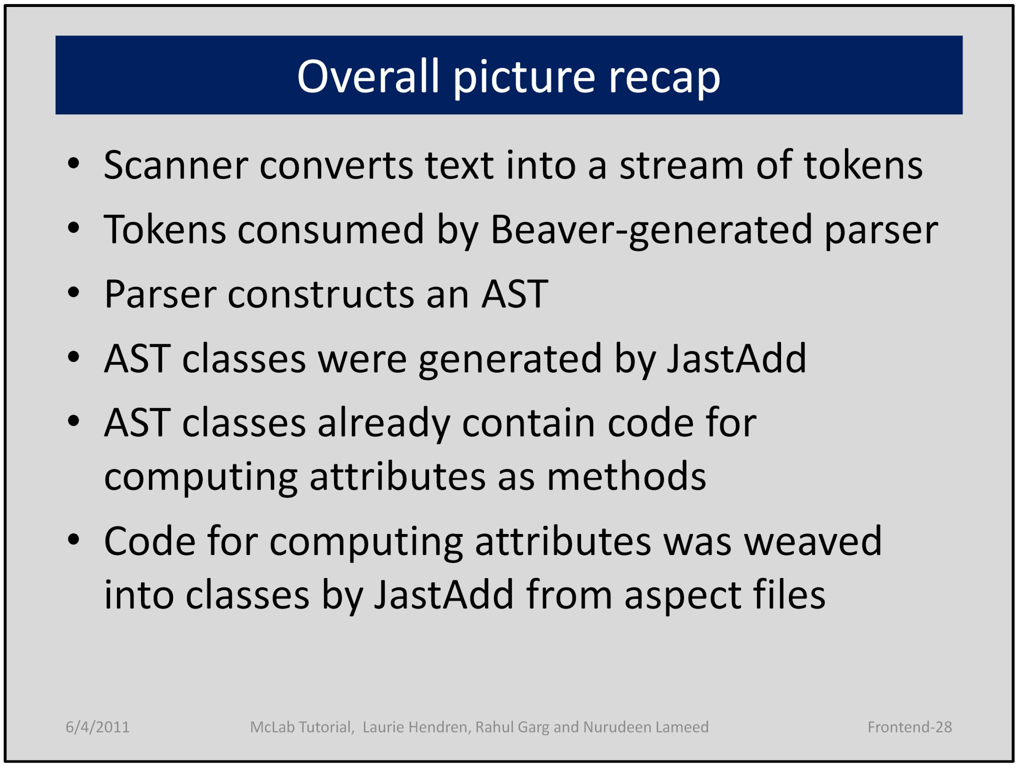
JastAdd

BinaryExpr.java

PlusExpr.java

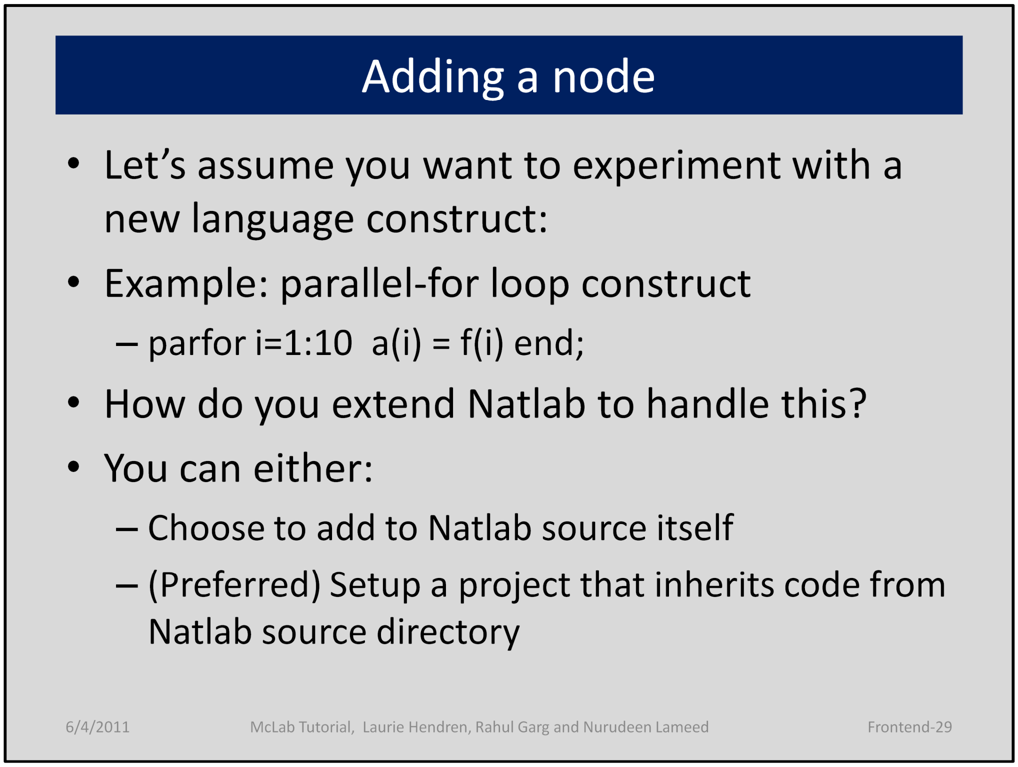
MinusExpr.java

6/4/2011 McLab Tutorial, Laurie Hendren, Rahul Garg and Nurudeen Lameed Frontend-27



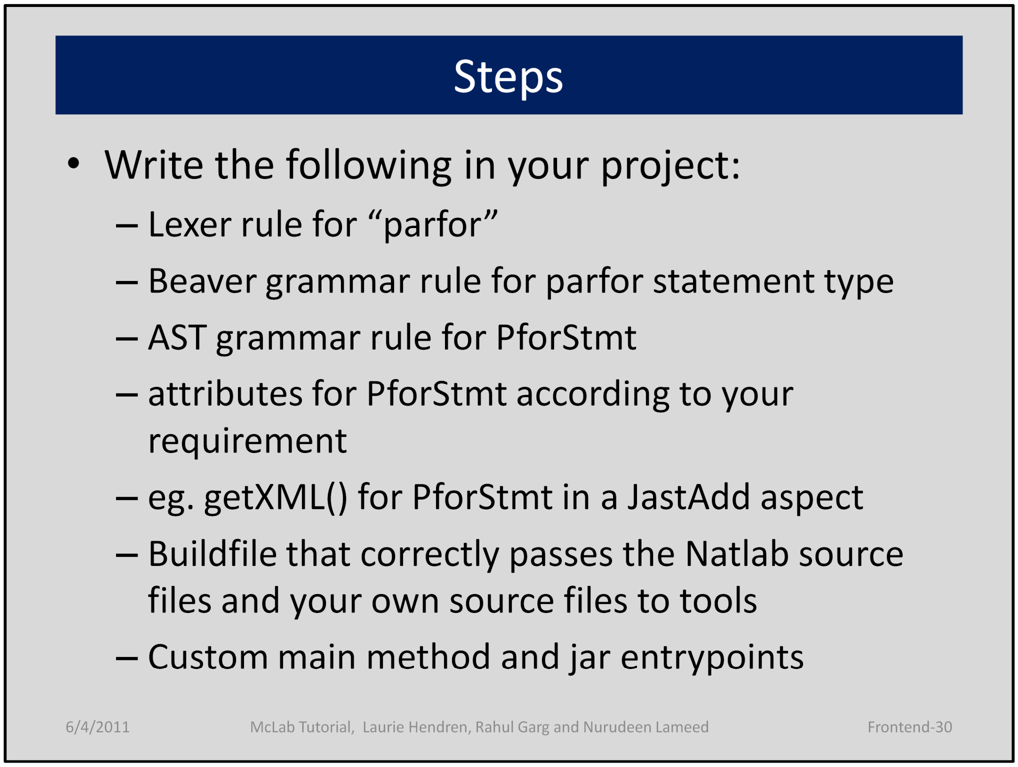
Overall picture recap

- Scanner converts text into a stream of tokens
- Tokens consumed by Beaver-generated parser
- Parser constructs an AST
- AST classes were generated by JastAdd
- AST classes already contain code for computing attributes as methods
- Code for computing attributes was weaved into classes by JastAdd from aspect files



Adding a node

- Let’s assume you want to experiment with a new language construct:
- Example: parallel-for loop construct
  - parfor i=1:10 a(i) = f(i) end;
- How do you extend Natlab to handle this?
- You can either:
  - Choose to add to Natlab source itself
  - (Preferred) Setup a project that inherits code from Natlab source directory



Steps

- Write the following in your project:
  - – Lexer rule for “parfor”
  - – Beaver grammar rule for parfor statement type
  - – AST grammar rule for PforStmt
  - – attributes for PforStmt according to your requirement
  - – eg. getXML() for PforStmt in a JastAdd aspect
  - – Buildfile that correctly passes the Natlab source files and your own source files to tools
  - – Custom main method and jar entrypoints