Cool AST

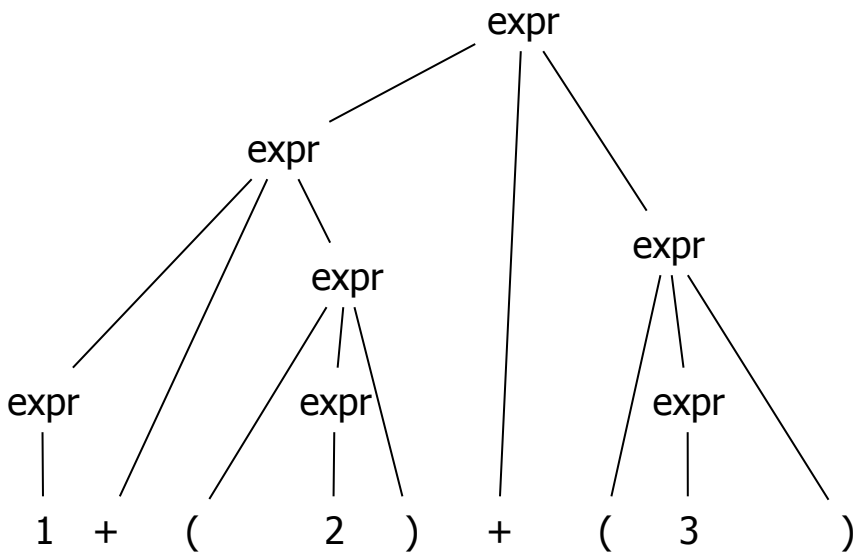
Greta Yorsh

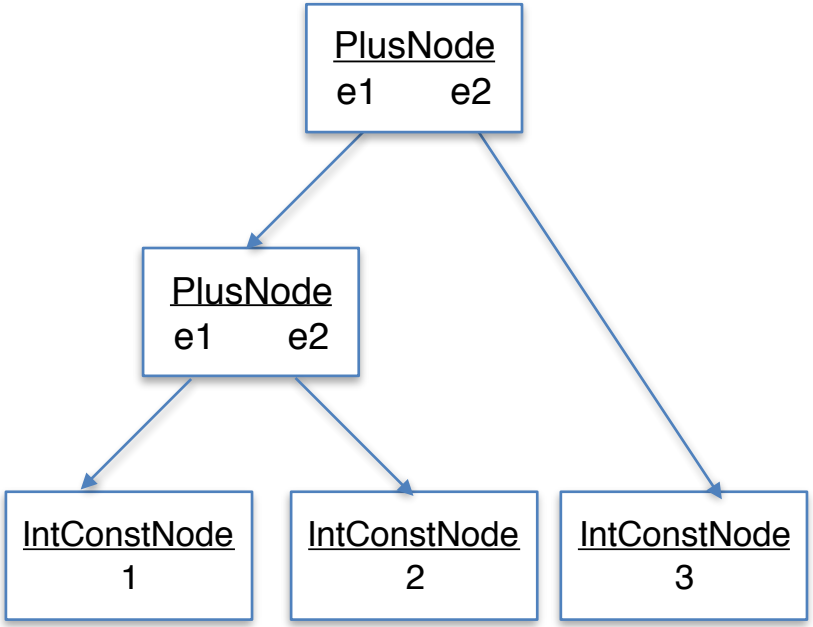
Cool AST

- Design
- Construction
- Traversal

Parse Tree vs Abstract Syntax Tree

```
grammar Example;
expr       : expr + expr | '(' expr ')' | IntConst;
IntConst : ('+'|'-')? [0-9]+ ;
WS          :  (' ' | '\n' |'\r' | '\t')+ -> skip
```





Abstract Syntax Tree (AST)

- A more useful representation of the parse tree
  - less clutter
  - actual level of details depends on your design
- Computation by AST traversal
  - debug printing
  - evaluation of expressions
- Basis for semantic analysis
- Later – annotate AST
  - type information
  - evaluation

(Most of) Cool Syntax

program ::= [*class*;]+

class ::= class TYPE [inherits TYPE] { [*feature*;]* }

feature ::= ID([*formal* [,*formal*]*]) : TYPE { *expr* }

ID : TYPE [<- *expr*]

formal ::= ID : TYPE

expr ::= ID <- *expr*

expr[@TYPE].ID([*expr* [, *expr*]*])

ID([*expr* [, *expr*]*])

if *expr* then *expr* else *expr* fi

while *expr* loop *expr* pool

{ [*expr*;]+ }

let ID : TYPE [<- *expr*] [,ID : TYPE [<- *expr*]]* in *expr*

case *expr* of [ID : TYPE => *expr*;]+ esac

new TYPE

isvoid *expr*

expr + *expr*

expr - *expr*

expr * *expr*

expr / *expr*

~*expr*

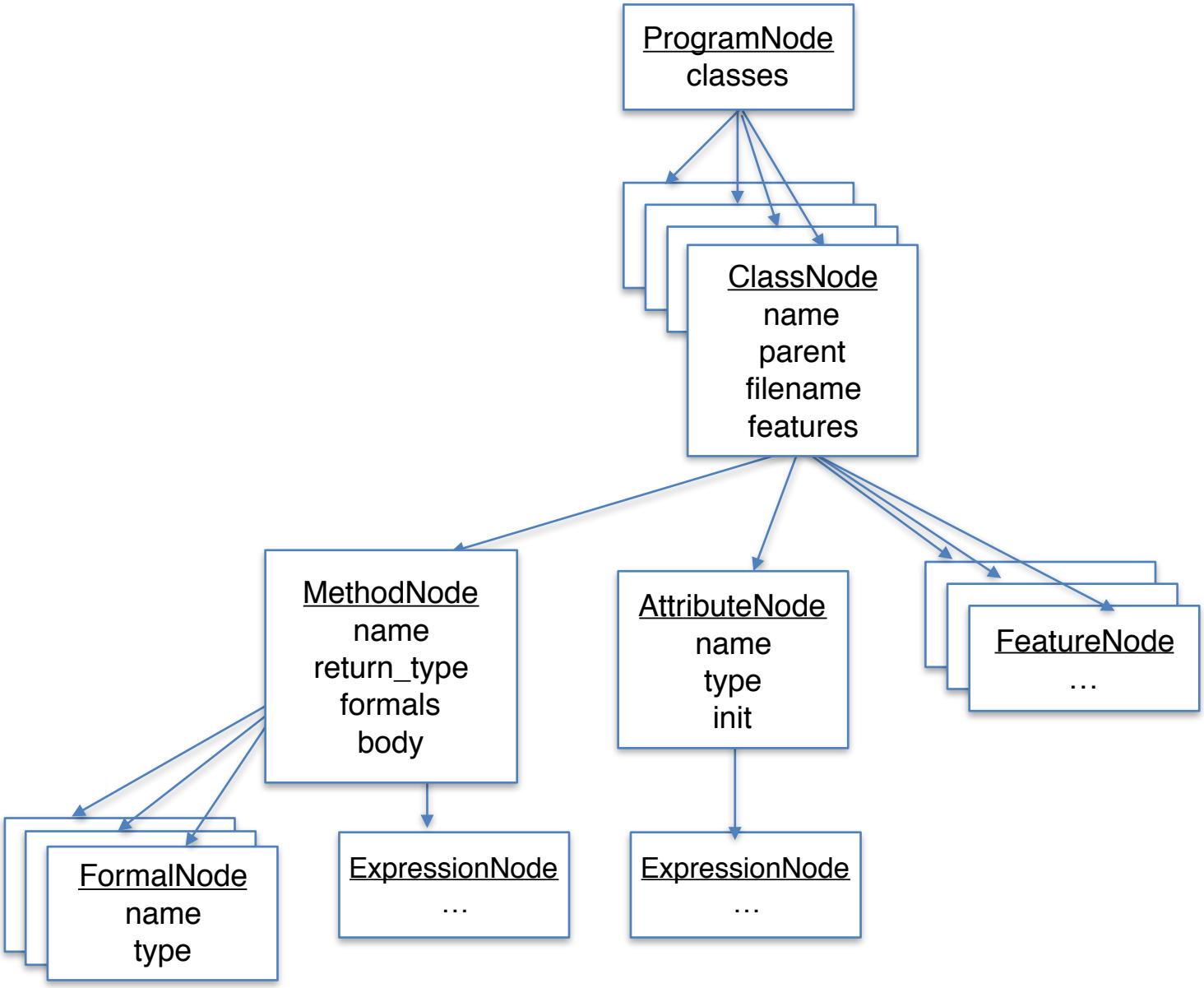
expr < *expr*

expr <= *expr*

expr = *expr*

...

Cool AST


ProgramNode
classes
ClassNode
name
parent
filename
features
MethodNode
name
return_type
formals
body
AttributeNode
name
type
init
FeatureNode
…
FormalNode
name
type
ExpressionNode
…
ExpressionNode
…

AST design: rules of thumb

- Abstract base class for AST nodes
- Class for the root of AST
- Abstract class for non-terminals with alternatives
- Class for each non-terminal or group of related non-terminals with similar functionality

AST design: rules of thumb

- Abstract base class for AST nodes
  - `abstract class TreeNode {…}`
- Class for the root of AST
  - `class ProgramNode extends TreeNode {…}`

Abstract class for non-terminals with alternative

- **feature : method | attribute;**

- `abstract class FeatureNode extends TreeNode {…}`
- `class MethodNode extends FeatureNode {…}`
- `class AttributeNode extends Feature {…}`

Abstract class for non-terminals with alternative

Abstract class for non-terminals with alternative

- **expr : ID ASSIGN expr l expr PLUS expr**
  …

Abstract class for non-terminals with alternative

- **expr : ID ASSIGN expr | expr PLUS expr ...**

- `abstract class ExpressionNode extends TreeNode`
- `class AssignNode extends ExpressionNode`
- `class PlusNode extends ExpressionNode`

Abstract class for non-terminals with alternative

- `abstract class FeatureNode extends TreeNode`
- `abstract class ExpressionNode extends TreeNode`
- `abstract class ConstNode extends ExpressionNode`
- `abstract class BinopNode extends ExpressionNode`
- `abstract class IntBinopNode extends BinopNode`
- `abstract class BoolBinopNode extends BinopNode`
- `abstract class IntUnopNode extends UnopNode`
- `abstract class BoolUnopNode extends UnopNode`

Class for non-terminals with similar functionality

- **class : CLASS TYPE (INHERITS TYPE)? ‘{‘ feature+ ‘}’;**

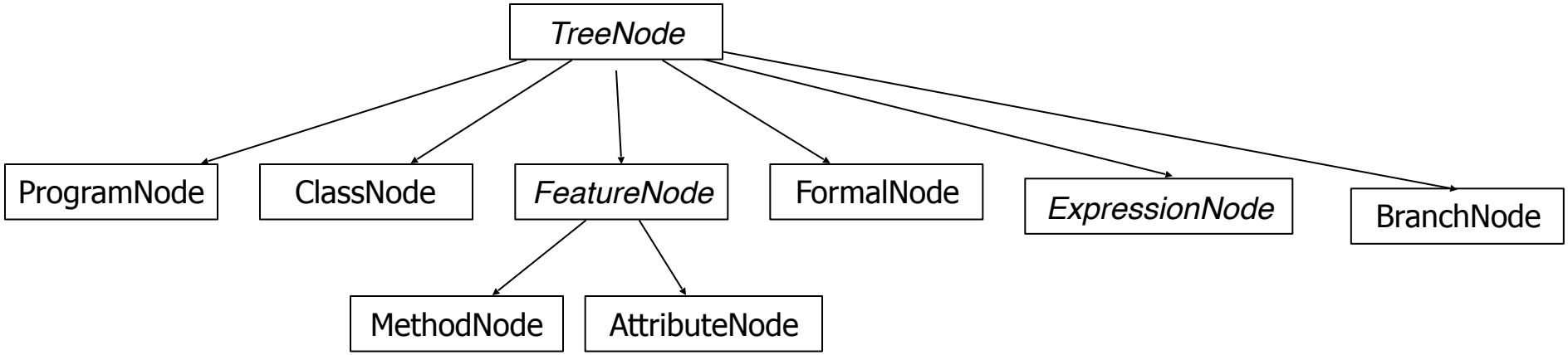
- `class ClassNode extends TreeNode {…}`

Class for non-terminals with similar functionality

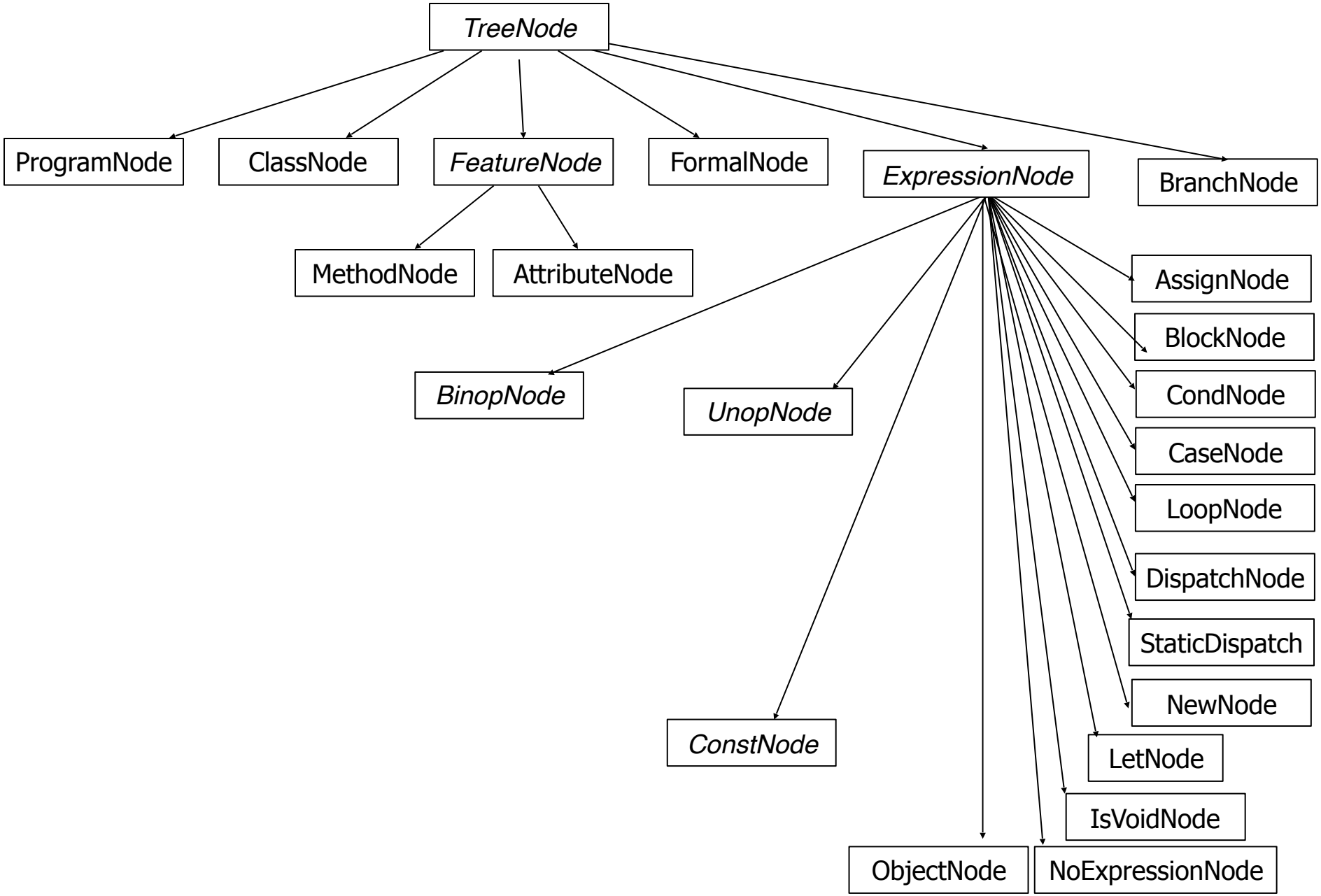
- **attr : ID ‘:’ TYPE (‘<-‘ expr)?**

- `class AttributeNode extends FeatureNode`

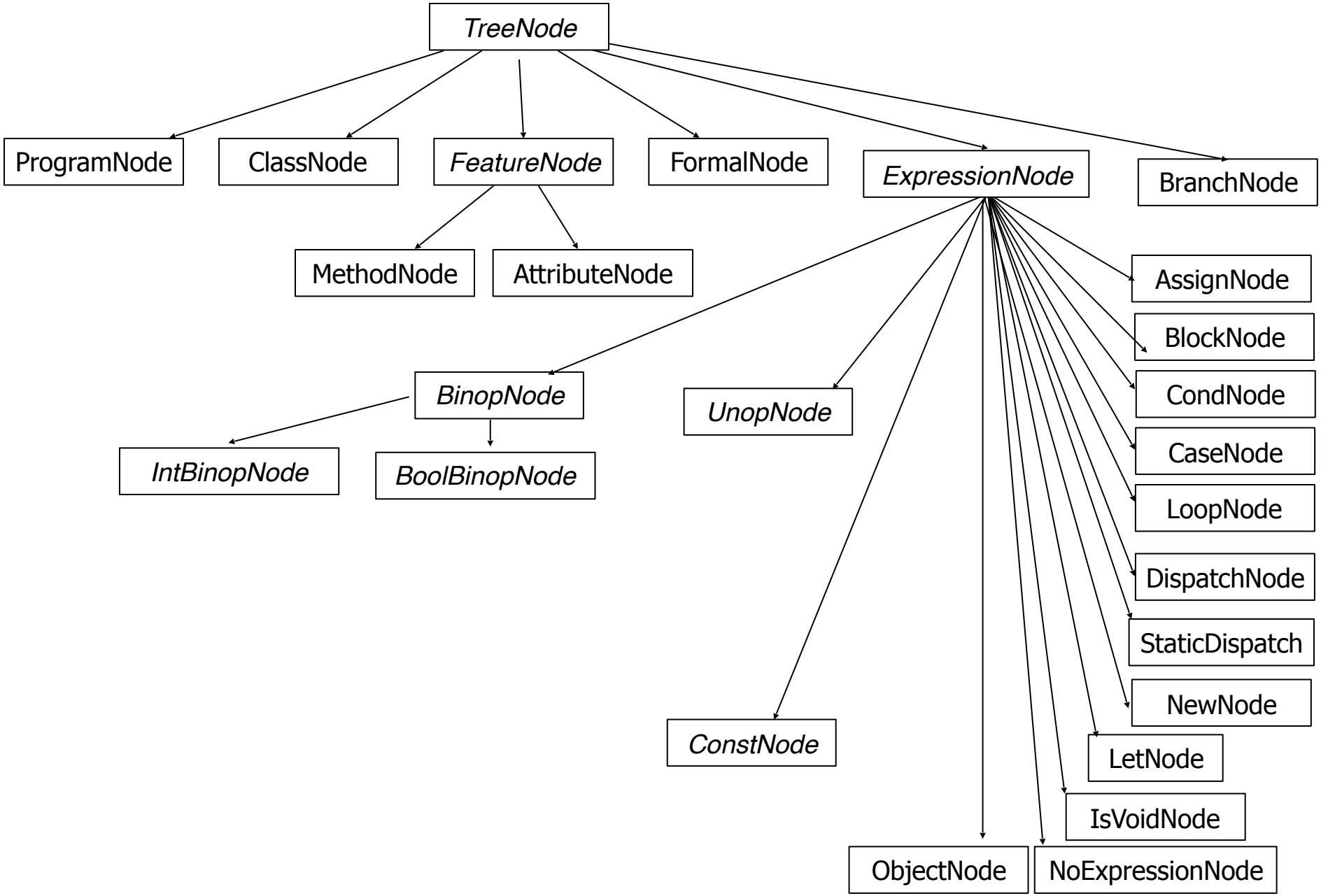
Class Hierarchy for Cool AST



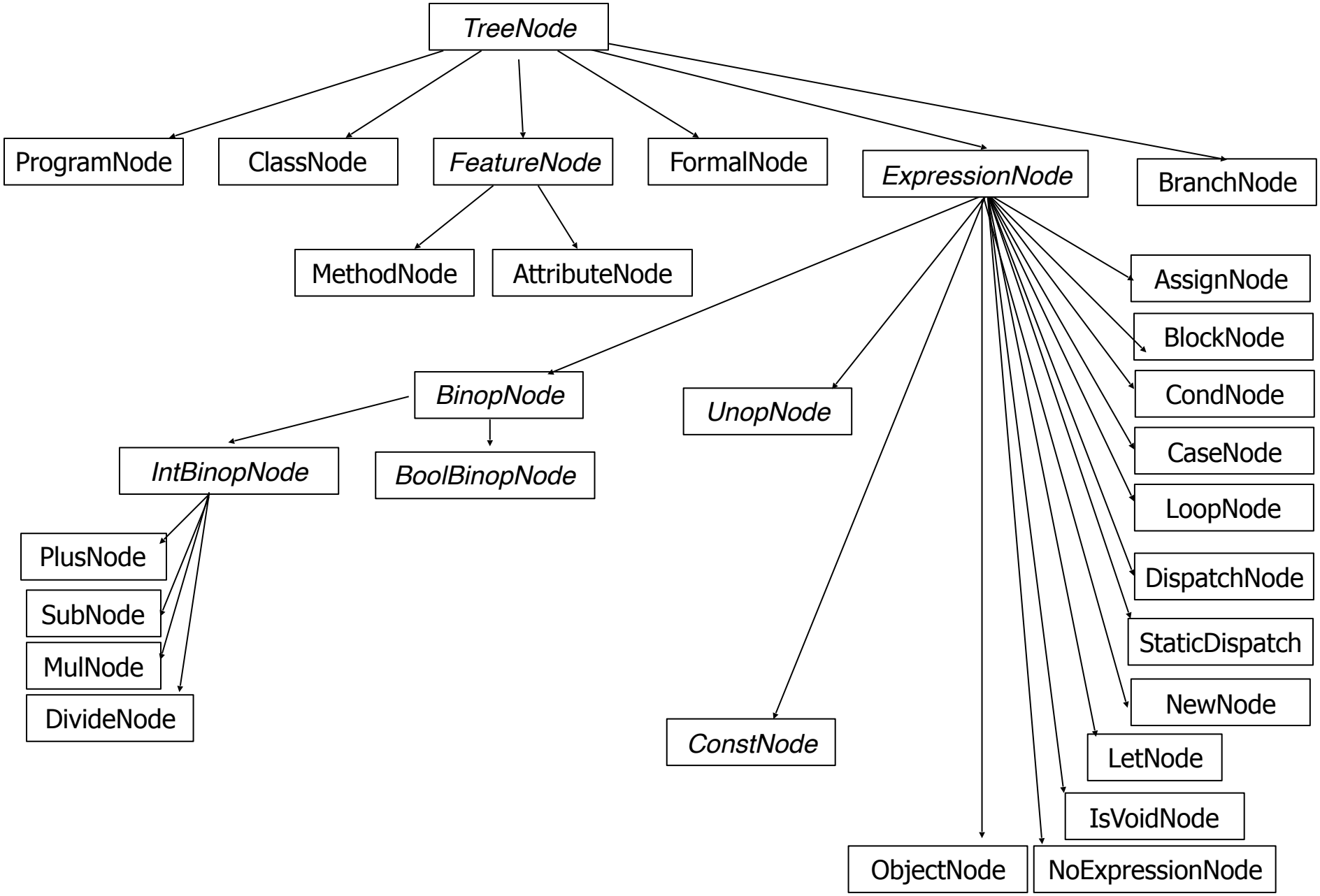
Class Hierarchy for Cool AST



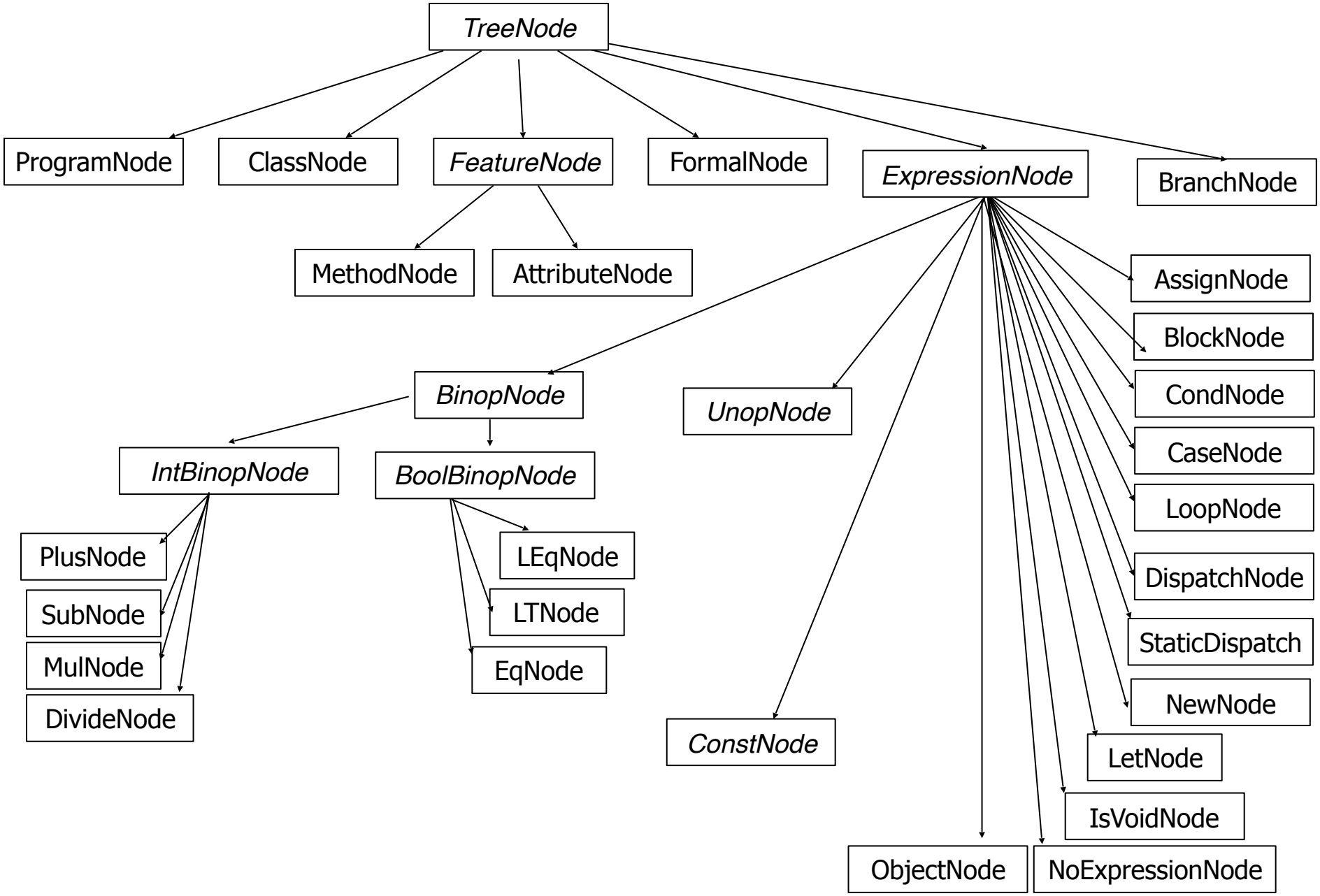
Class Hierarchy for Cool AST



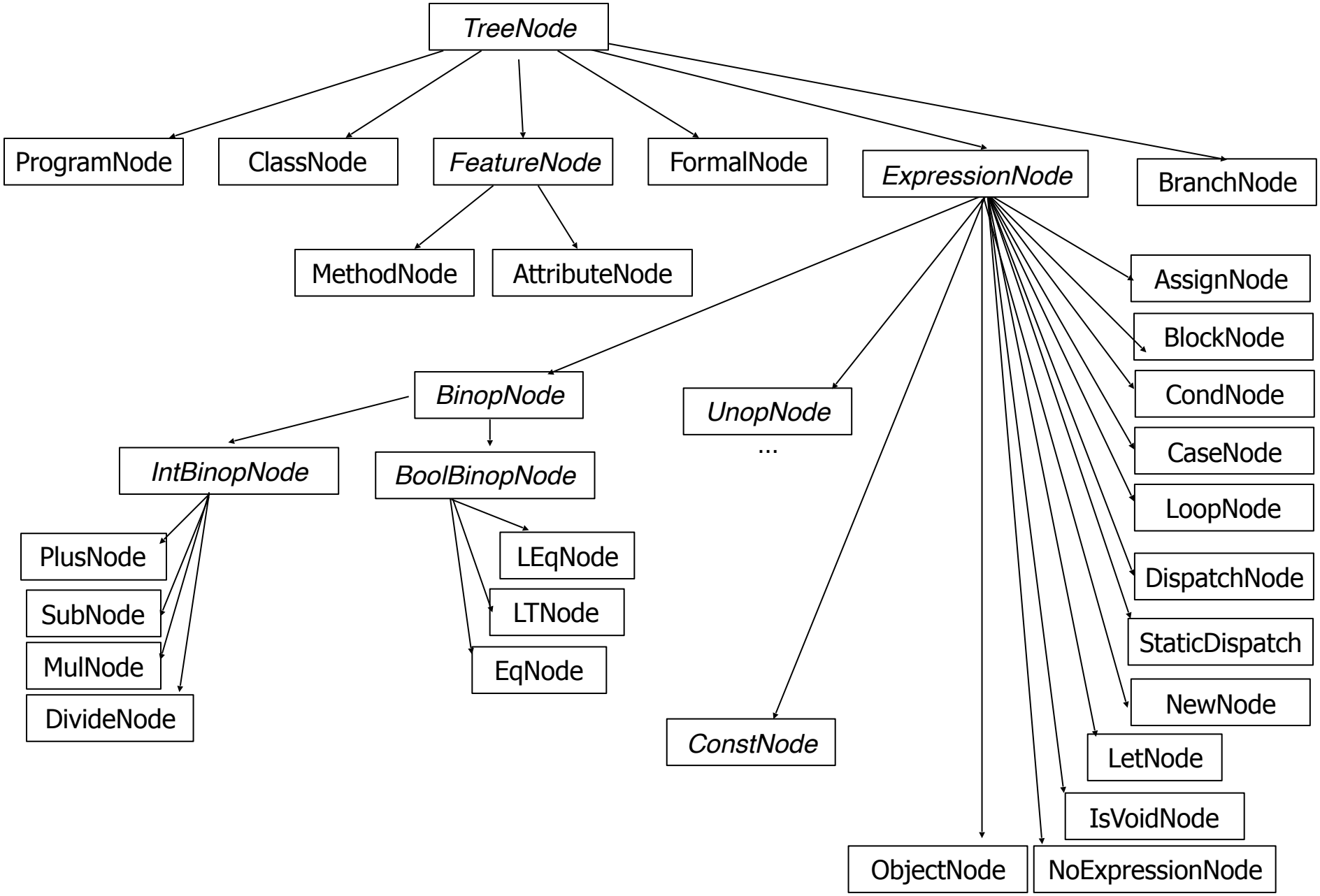
Class Hierarchy for Cool AST



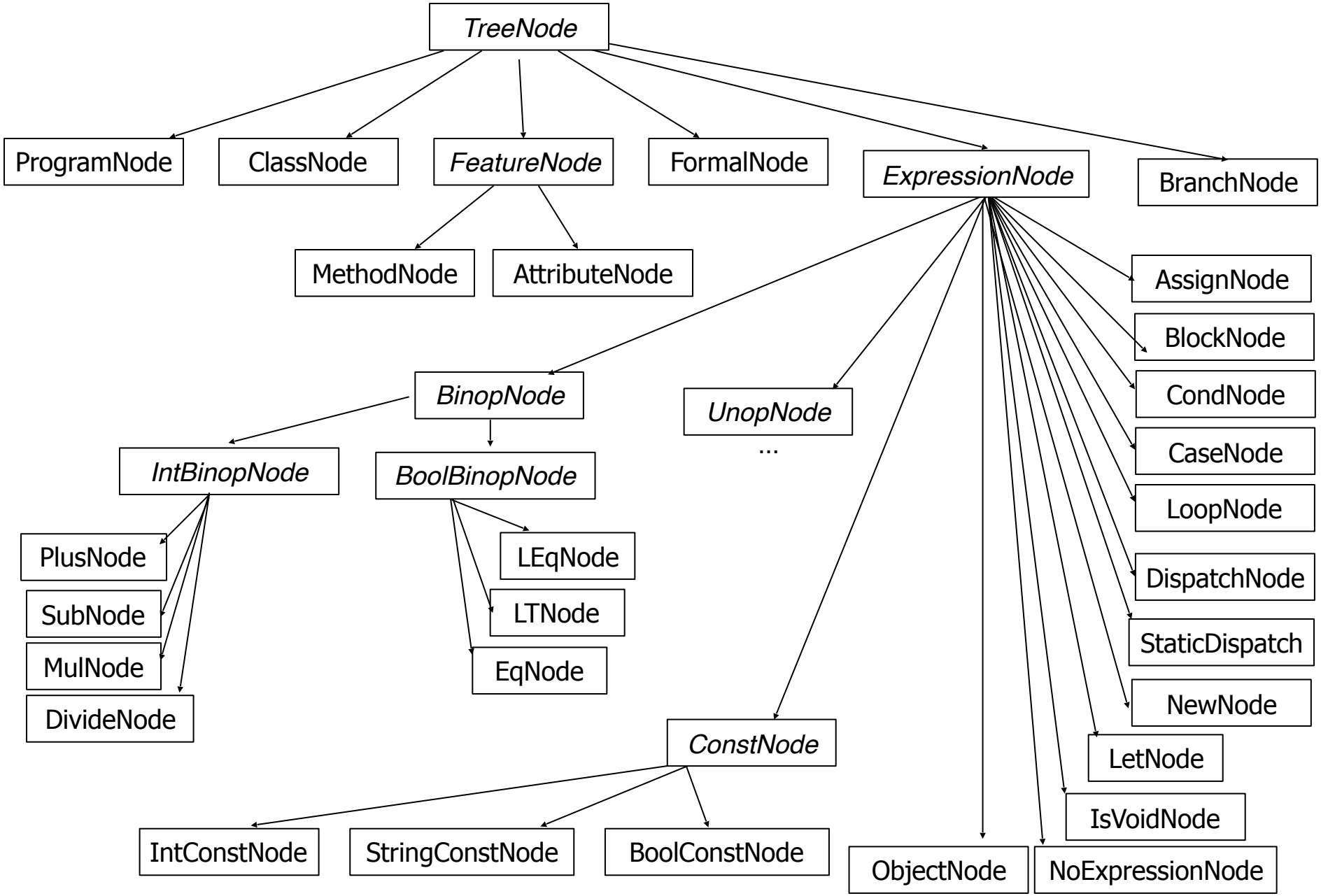
Class Hierarchy for Cool AST



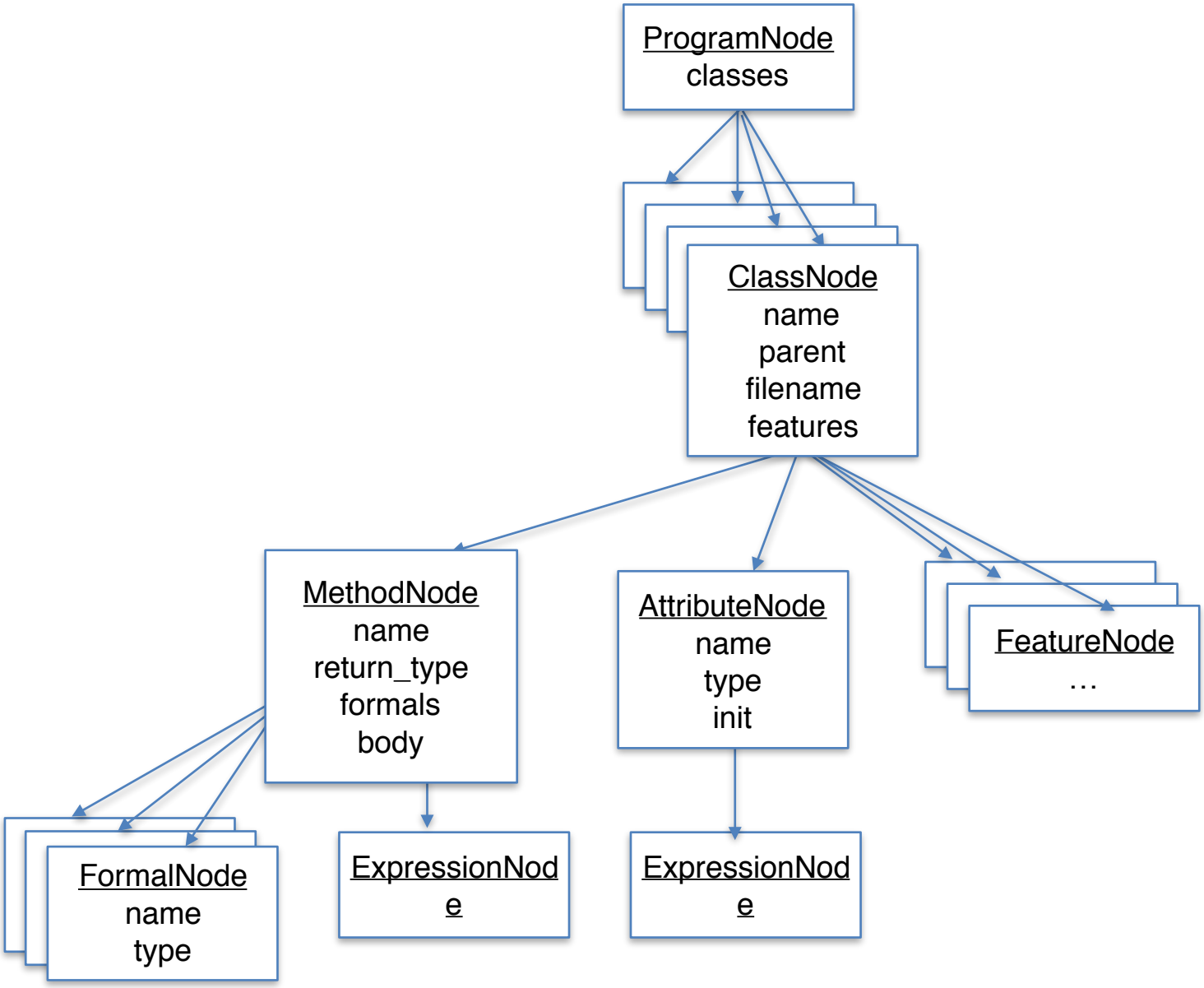
Class Hierarchy for Cool AST



Class Hierarchy for Cool AST



Cool AST


ProgramNode
classes
ClassNode
name
parent
filename
features
MethodNode
name
return_type
formals
body
AttributeNode
name
type
init
FeatureNode
…
FormalNode
name
type
ExpressionNode
ExpressionNode

AST construction: approaches

- When?
  - during parsing (execute on each production)
  - after parsing by traversing the parse tree
- How?
  - actions
    - part of parser generation specification
    - grammar description mixed with code
  - listener or visitor
    - separate grammar from code

AST traversal

- Examples operations
  - pretty printing
  - type checking
  - register allocation
  - code generation
- How?
  - naive: a method per operation in each node class
  - visitor design pattern

Visitor Design Pattern

- Separate object **representation** from **operations** on objects of a data structure
- Each operation implemented as separate visitor

- Java: visitor implements double dispatch

- works well when representation is fixed

AST Visitor

AST Visitor

```
interface TreeVisitor {
  void visit(Tree n);
  void visit(TreeNode n);
  void visit(ProgramNode n);
  void visit(ClassNode n);
  void visit(MethodNode n);
  void visit(BlockNode n);
  void visit(AssignNode n);
  …
}
```

AST Visitor

```
interface TreeVisitor {
  void visit(Tree n);
  void visit(TreeNode n);
  void visit(ProgramNode n);
  void visit(ClassNode n);
  void visit(MethodNode n);
  void visit(BlockNode n);
  void visit(AssignNode n);
  …
}
```

```
interface Tree {
   void accept(TreeVisitor visitor);
}
```

AST Visitor

```
interface TreeVisitor {
  void visit(Tree n);
  void visit(TreeNode n);
  void visit(ProgramNode n);
  void visit(ClassNode n);
  void visit(MethodNode n);
  void visit(BlockNode n);
  void visit(AssignNode n);
  …
}
```

```
interface Tree {
   void accept(TreeVisitor visitor);
}
```

```
abstract class TreeNode implements Tree {
   void accept(TreeVisitor visitor){
      v.visit(this);
   }
}
```

AST Visitor

```
interface TreeVisitor {
  void visit(Tree n);
  void visit(TreeNode n);
  void visit(ProgramNode n);
  void visit(ClassNode n);
  void visit(MethodNode n);
  void visit(BlockNode n);
  void visit(AssignNode n);
  …
}
```

```
class BaseVisitor implements TreeVisitor {
  void visit(Tree n){…}
  void visit(TreeNode n) {…}
  void visit(ProgramNode n) {
    foreach (ClassNode c : n.getClasses()
      c.accept(v);
  }
  void visit(BinopNode n) {
     n.getE1().accept(v);
     n.getE2().accept(v);
  }…
}
```

```
interface Tree {
   void accept(TreeVisitor visitor);
}
```

```
abstract class TreeNode implements Tree {
   void accept(TreeVisitor visitor){
      v.visit(this);
   }
}
```

AST Visitor

```
interface TreeVisitor {
  void visit(Tree n);
  void visit(TreeNode n);
  void visit(ProgramNode n);
  void visit(ClassNode n);
  void visit(MethodNode n);
  void visit(BlockNode n);
  void visit(AssignNode n);
  …
}
```

```
class BaseVisitor implements TreeVisitor {
  void visit(Tree n){…}
  void visit(TreeNode n) {…}
  void visit(ProgramNode n) {
    foreach (ClassNode c : n.getClasses()
      c.accept(v);
  }
  void visit(BinopNode n) {
     n.getE1().accept(v);
     n.getE2().accept(v);
  }…
}
```

```
interface Tree {
   void accept(TreeVisitor visitor);
}
```

```
abstract class TreeNode implements Tree {
   void accept(TreeVisitor visitor){
      v.visit(this);
   }
}
```

```
class DumpVisitor
   extends BaseVisitor {
  void visit(ClassNode n){
     out.println("_class");
     super.visit(n);
  } …
  void visit(ConstNode n){
     out.print(n.getVal());
  }
  …
}
```

AST Visitor

```
interface TreeVisitor {
  void visit(Tree n);
  void visit(TreeNode n);
  void visit(ProgramNode n);
  void visit(ClassNode n);
  void visit(MethodNode n);
  void visit(BlockNode n);
  void visit(AssignNode n);
  …
}
```

```
class BaseVisitor implements TreeVisitor {
  void visit(Tree n){…}
  void visit(TreeNode n) {…}
  void visit(ProgramNode n) {
    foreach (ClassNode c : n.getClasses()
      c.accept(v);
  }
  void visit(BinopNode n) {
     n.getE1().accept(v);
     n.getE2().accept(v);
  }…
}
```

```
interface Tree {
   void accept(TreeVisitor visitor);
}
```

```
abstract class TreeNode implements Tree {
   void accept(TreeVisitor visitor){
      v.visit(this);
   }
}
```

```
void main() {
   Tree root; …
   TreeVisitor v = new DumpVisitor ();
   root.accept(v); …
```

```
class DumpVisitor
   extends BaseVisitor {
  void visit(ClassNode n){
     out.println("_class");
     super.visit(n);
  } …
  void visit(ConstNode n){
     out.print(n.getVal());
  }
  …
}
```

Visitor with return value

Visitor with return value

```
class EvalVisitor implements BaseVisitor {
  Object visit(PlusNode n){
    Integer v1 = (Integer) n.getE1().accept();
    Integer v2 = (Integer) n.getE2().accept();
    return v1 + v2;
  }
  Object visit(IntConstNode n){
    return (Integer) n.getVal();
 }
}…
```

Visitor with return value

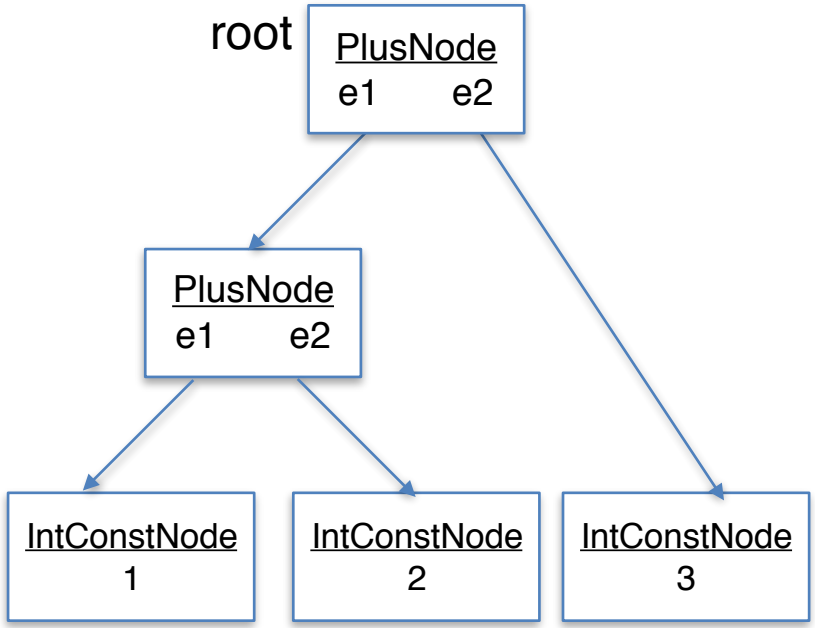
```
class EvalVisitor implements BaseVisitor {
  Object visit(PlusNode n){
    Integer v1 = (Integer) n.getE1().accept();
    Integer v2 = (Integer) n.getE2().accept();
    return v1 + v2;
  }
  Object visit(IntConstNode n){
    return (Integer) n.getVal();
 }
}…
```

```
void main() {
   Tree root; …
   EvalVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v);
   …
}
```

Visitor with return value

```
class EvalVisitor implements BaseVisitor {
  Object visit(PlusNode n){
    Integer v1 = (Integer) n.getE1().accept();
    Integer v2 = (Integer) n.getE2().accept();
    return v1 + v2;
  }
  Object visit(IntConstNode n){
    return (Integer) n.getVal();
 }
}…
```

```
void main() {
   Tree root; …
   EvalVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v);
   …
}
```



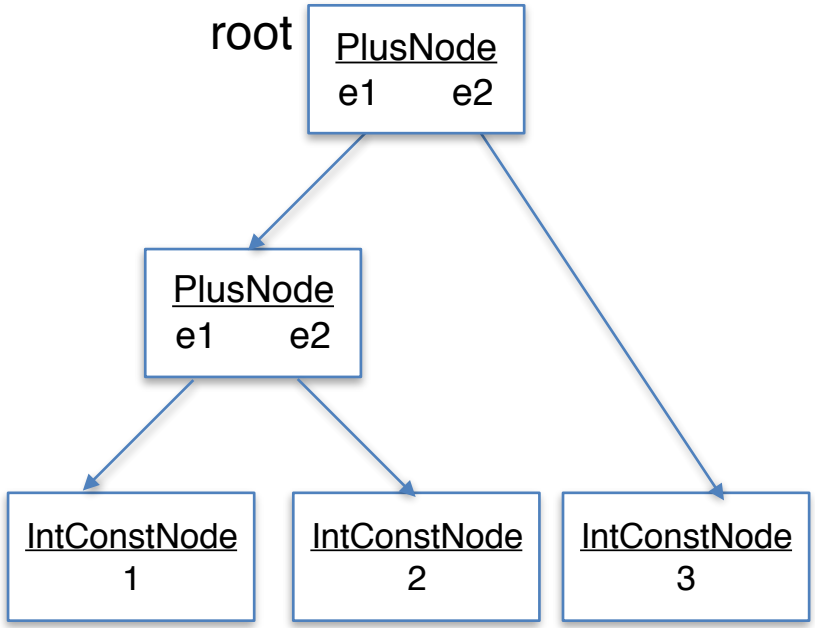
Visitor with return value

```
class IntConstNode extends ConstNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class PlusNode extends IntBinopNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class EvalVisitor implements BaseVisitor {
  Object visit(PlusNode n){
    Integer v1 = (Integer) n.getE1().accept();
    Integer v2 = (Integer) n.getE2().accept();
    return v1 + v2;
  }
  Object visit(IntConstNode n){
    return (Integer) n.getVal();
 }
}…
```

```
void main() {
   Tree root; …
   EvalVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v);
   …
}
```



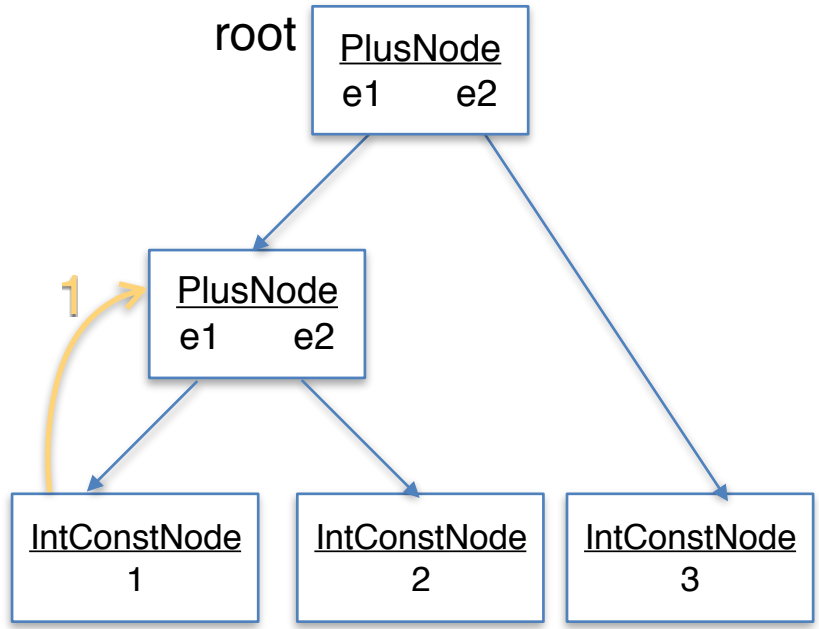
Visitor with return value

```
class IntConstNode extends ConstNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class PlusNode extends IntBinopNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class EvalVisitor implements BaseVisitor {
  Object visit(PlusNode n){
    Integer v1 = (Integer) n.getE1().accept();
    Integer v2 = (Integer) n.getE2().accept();
    return v1 + v2;
  }
  Object visit(IntConstNode n){
    return (Integer) n.getVal();
 }
}…
```

```
void main() {
   Tree root; …
   EvalVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v);
   …
}
```



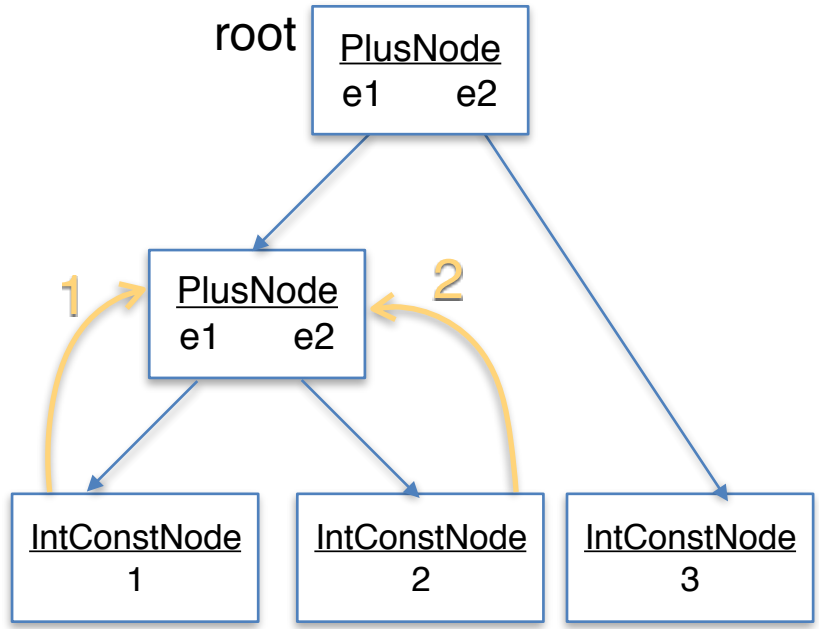
Visitor with return value

```
class IntConstNode extends ConstNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class PlusNode extends IntBinopNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class EvalVisitor implements BaseVisitor {
  Object visit(PlusNode n){
    Integer v1 = (Integer) n.getE1().accept();
    Integer v2 = (Integer) n.getE2().accept();
    return v1 + v2;
  }
  Object visit(IntConstNode n){
    return (Integer) n.getVal();
 }
}…
```

```
void main() {
   Tree root; …
   EvalVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v);
   …
}
```



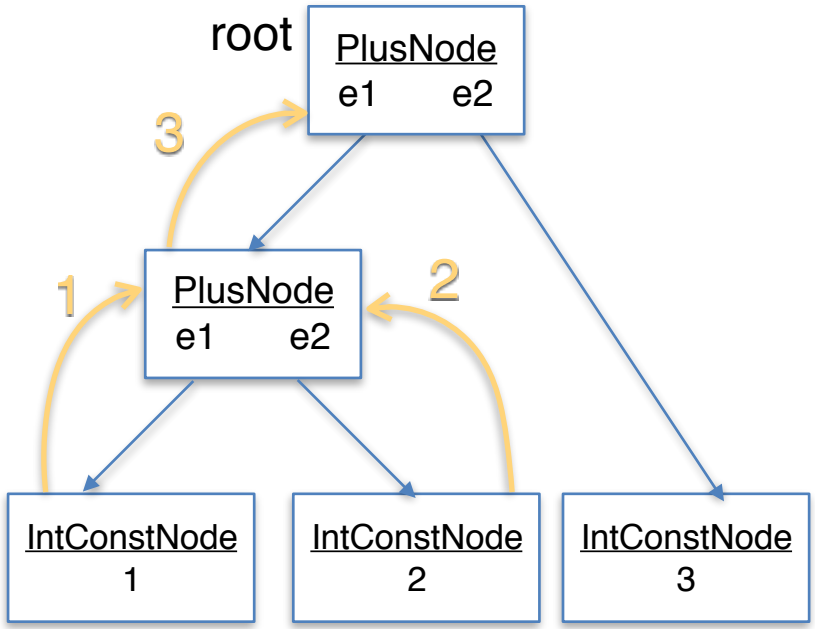
Visitor with return value

```
class IntConstNode extends ConstNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class PlusNode extends IntBinopNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class EvalVisitor implements BaseVisitor {
  Object visit(PlusNode n){
    Integer v1 = (Integer) n.getE1().accept();
    Integer v2 = (Integer) n.getE2().accept();
    return v1 + v2;
  }
  Object visit(IntConstNode n){
    return (Integer) n.getVal();
 }
}…
```

```
void main() {
   Tree root; …
   EvalVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v);
   …
}
```



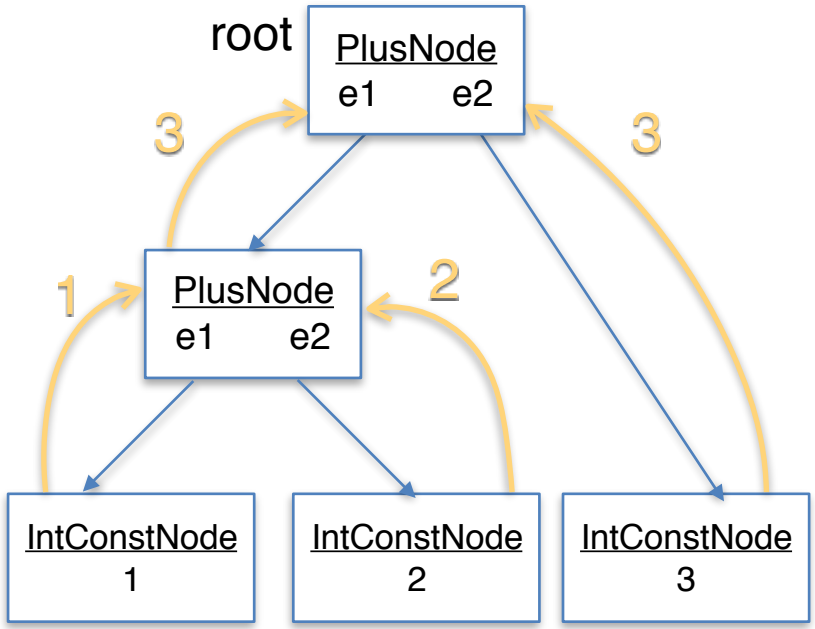
Visitor with return value

```
class IntConstNode extends ConstNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class PlusNode extends IntBinopNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class EvalVisitor implements BaseVisitor {
  Object visit(PlusNode n){
    Integer v1 = (Integer) n.getE1().accept();
    Integer v2 = (Integer) n.getE2().accept();
    return v1 + v2;
  }
  Object visit(IntConstNode n){
    return (Integer) n.getVal();
 }
}…
```

```
void main() {
   Tree root; …
   EvalVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v);
   …
}
```



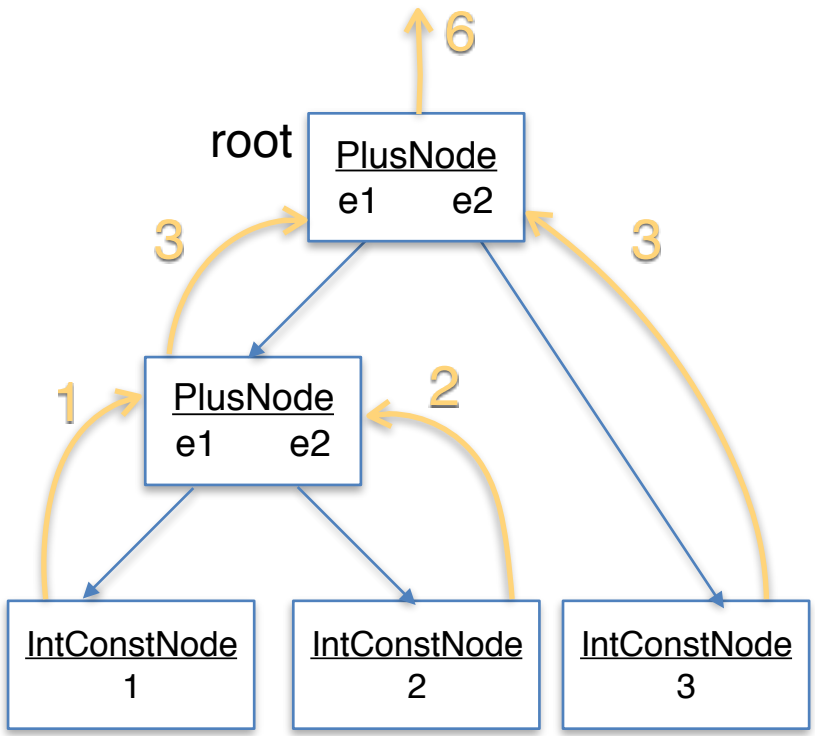
Visitor with return value

```
class IntConstNode extends ConstNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class PlusNode extends IntBinopNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }
}
```

```
class EvalVisitor implements BaseVisitor {
  Object visit(PlusNode n){
    Integer v1 = (Integer) n.getE1().accept();
    Integer v2 = (Integer) n.getE2().accept();
    return v1 + v2;
  }
  Object visit(IntConstNode n){
    return (Integer) n.getVal();
 }
}…
```

```
void main() {
   Tree root; …
   EvalVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v);
   …
}
```



Visitor variations

```
public class IntConstNode extends ConstNode {
   public Object accept(TreeVisitor visitor, Object data){
      return v.visit(this);
   }
}
```

```
public class PlusNode extends IntBinopNode {
   public Object accept(TreeVisitor visitor, Object data){
      return v.visit(this);
   }
}
```

```
class EvalVisitor implements TreeVisitor{
public:
    Object visit(ClassNode e, Object o);
    Object visit(MethodNode e , Object o);
    Object visit(BlockNode e , Object o);
    Object visit(AssignNode e , Object o);
    ...
}
```

```
void main() {
   Tree root; …
   TreeVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v,data);
   …
}
```

Propagate values
down the AST and back

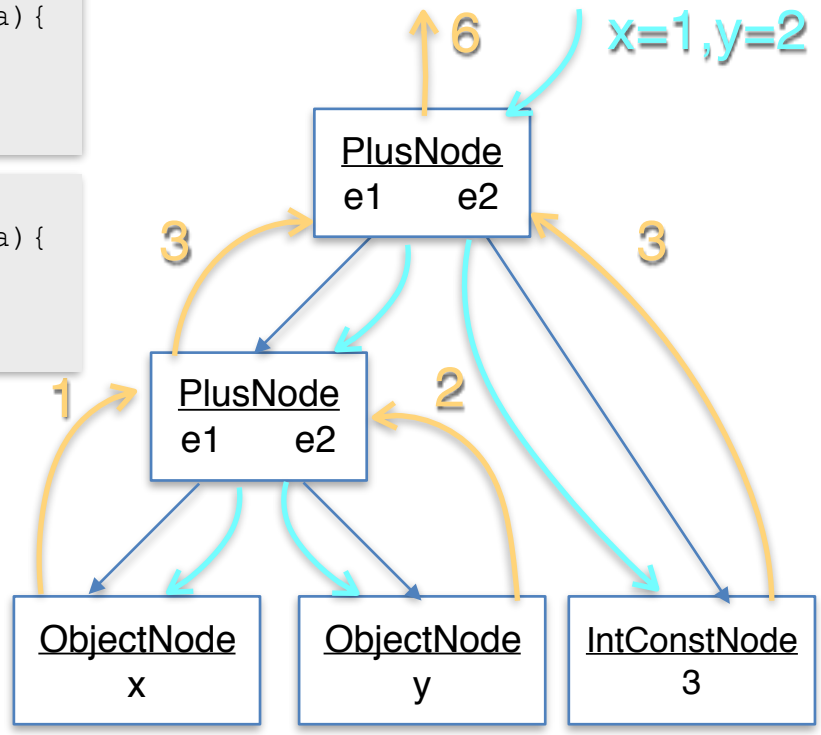
Visitor variations

```
public class IntConstNode extends ConstNode {
   public Object accept(TreeVisitor visitor, Object data){
      return v.visit(this);
   }
}
```

```
public class PlusNode extends IntBinopNode {
   public Object accept(TreeVisitor visitor, Object data){
      return v.visit(this);
   }
}
```

```
class EvalVisitor implements TreeVisitor{
public:
    Object visit(ClassNode e, Object o);
    Object visit(MethodNode e , Object o);
    Object visit(BlockNode e , Object o);
    Object visit(AssignNode e , Object o);
    ...
}
```

```
void main() {
   Tree root; …
   TreeVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v,data);
   …
}
```



Propagate values
down the AST and back

Visitors: which method is called?

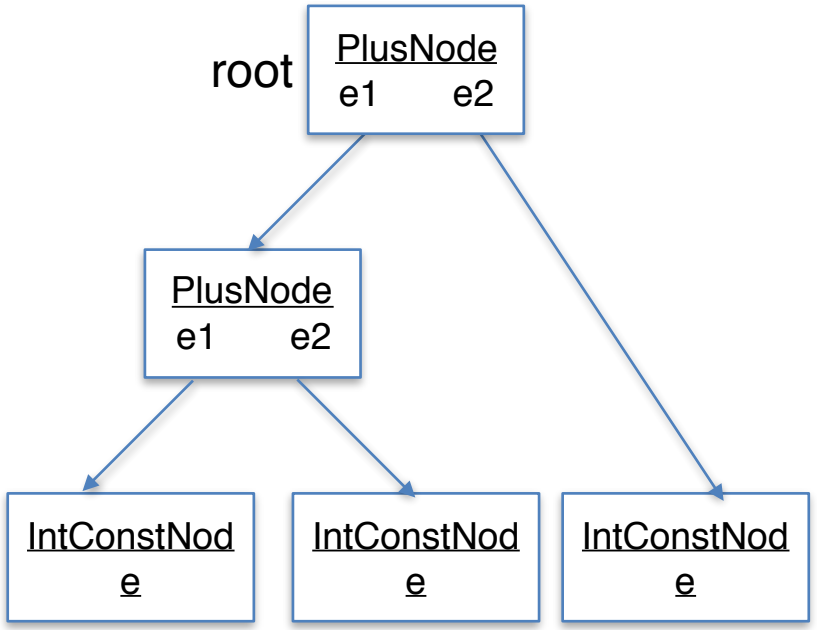
- static dispatch
- dynamic dispatch
  - single
  - multiple

- method overloading
- method overriding

Single dispatch

```
class EvalVisitor implements BaseVisitor {
  public Object visit(PlusNode n){
    Integer v1 = (Integer) visit(n.getE1());
    Integer v2 = (Integer) visit(n.getE2());
    return v1 + v2;
  }
  public Object visit(IntConstNode n){
    return (Integer) n.getVal();
  }
  public Object visit(ExpressionNode n){
    return 0;
  }
  …
}
```

```
void main() {
   PlusNode root; …
   TreeVisitor v = new EvalVisitor ();
   Integer result = (Integer) v.visit(root);
   …
}
```



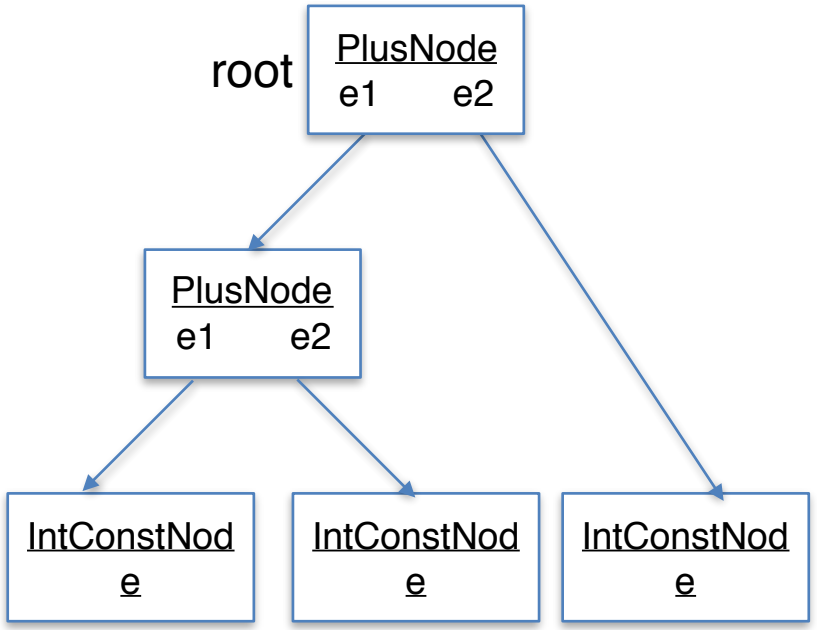
Double dispatch

```
class EvalVisitor implements BaseVisitor {
  public Object visit(PlusNode n){
    Integer v1 = (Integer) n.getE1().accept();
    Integer v2 = (Integer) n.getE2().accept();
    return v1 + v2;
  }
  public Object visit(IntConstNode n){
    return (Integer) n.getVal();
  }
  public Object visit(ExpressionNode n){
    return 0;
  }
  …
}
```

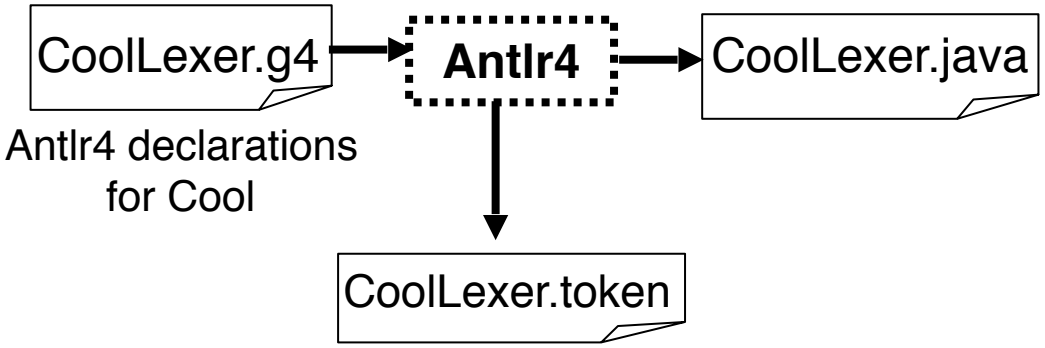
```
class PlusNode extends IntBinopNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }…
}
```

```
class IntConstNode extends ConstNode {
   Object accept(TreeVisitor visitor){
      return v.visit(this);
   }…
}
```

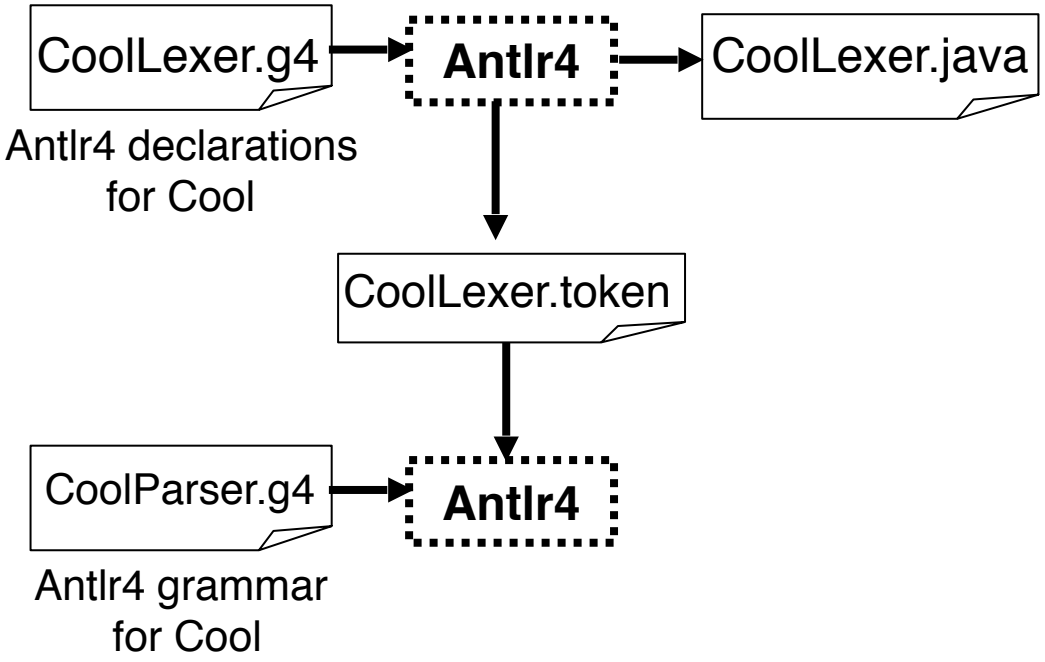
```
void main() {
   Tree root; …
   TreeVisitor v = new EvalVisitor ();
   Integer result = (Integer) root.accept(v);
   …
}
```



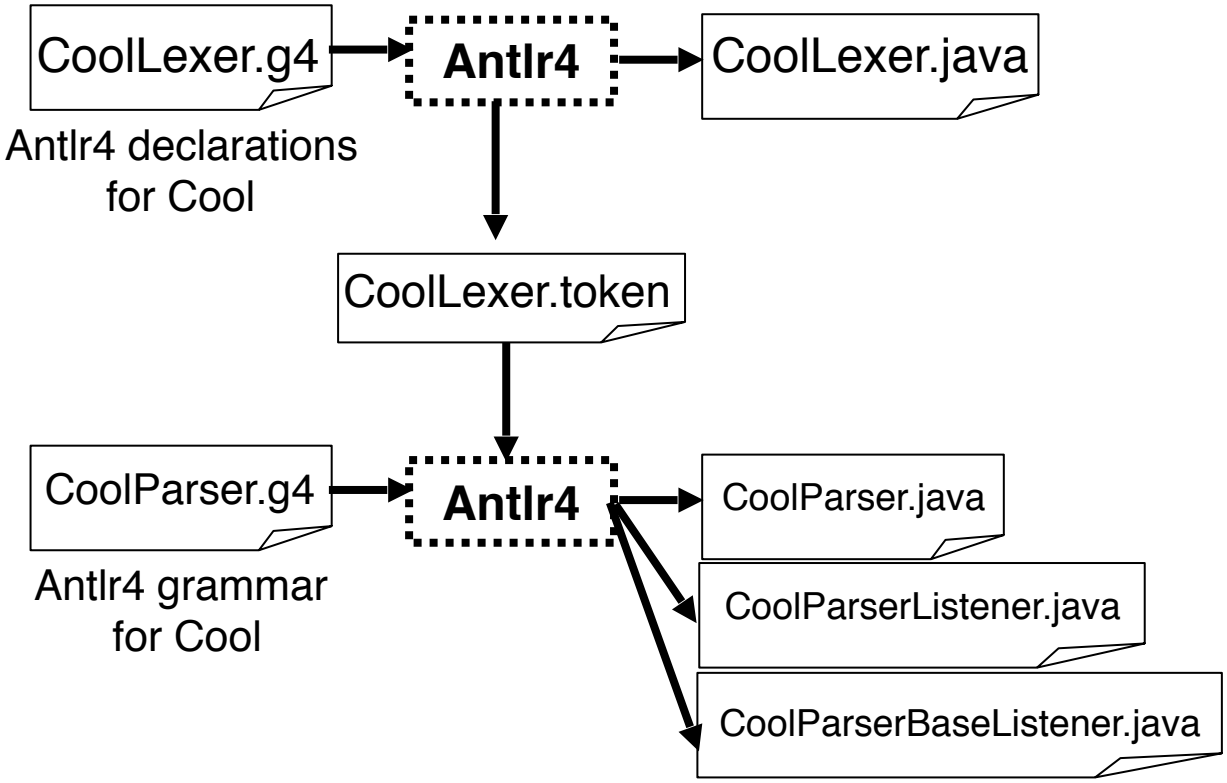
Antlr: build and run a parser



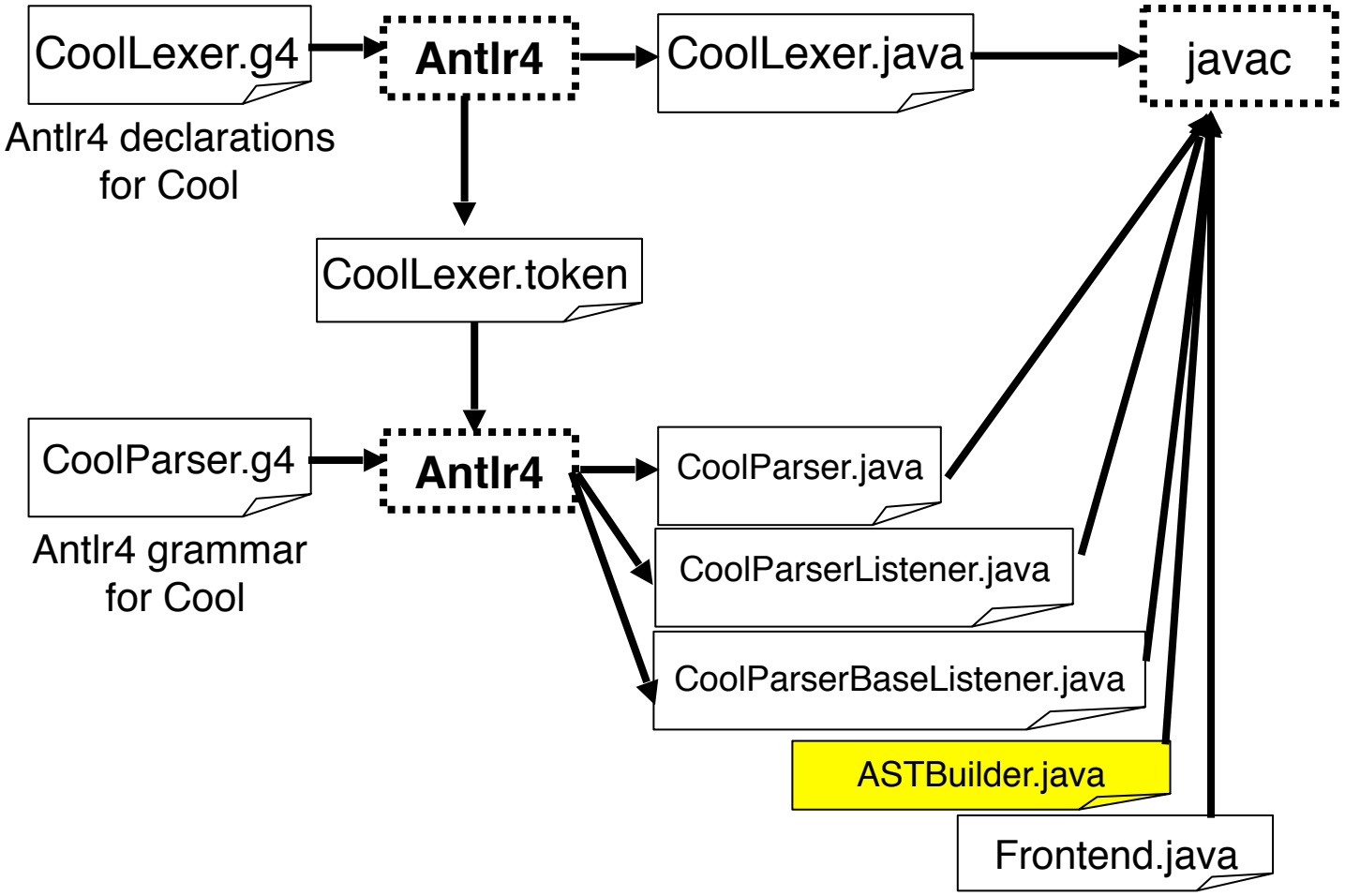
Antlr: build and run a parser



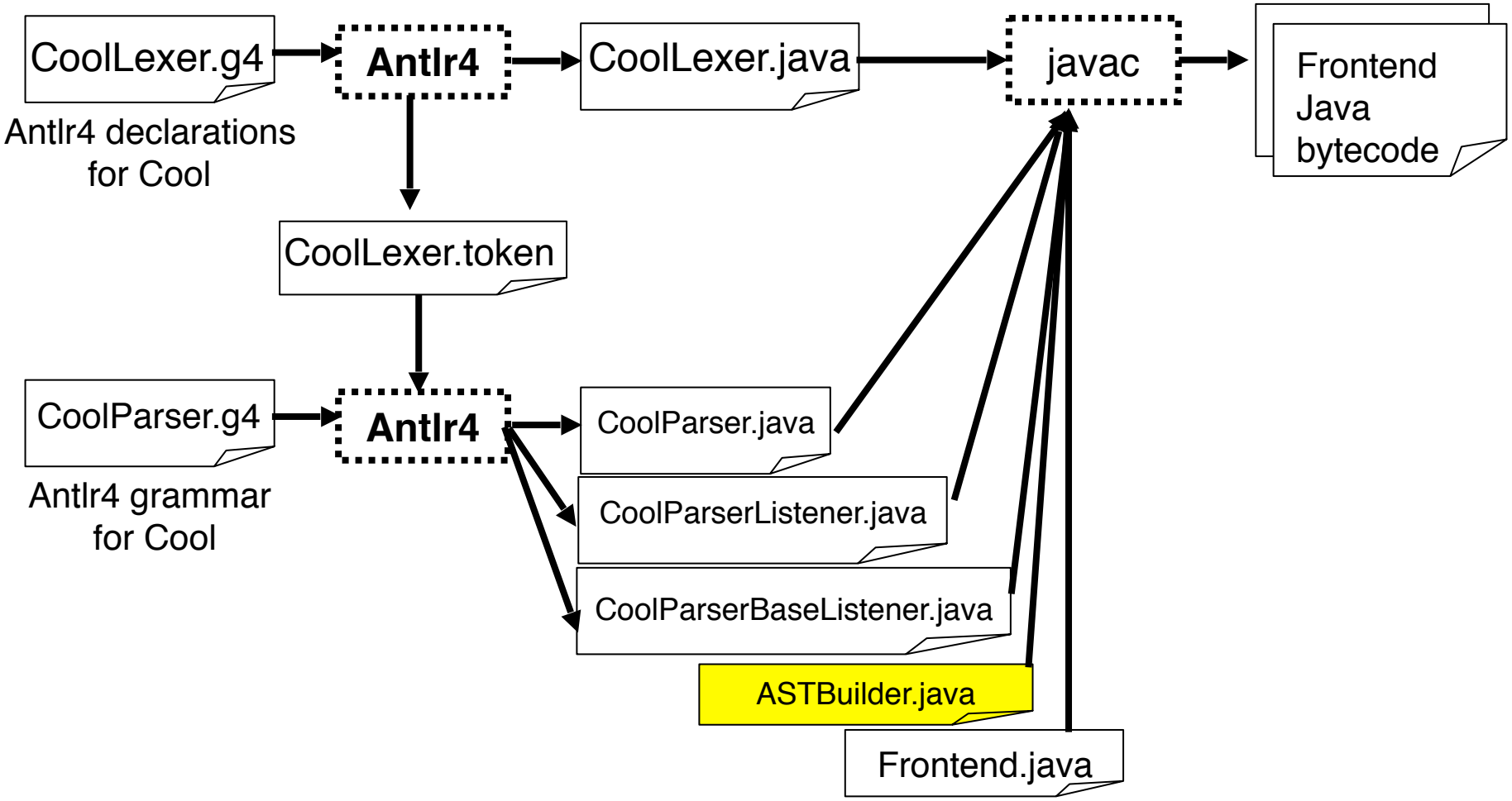
Antlr: build and run a parser



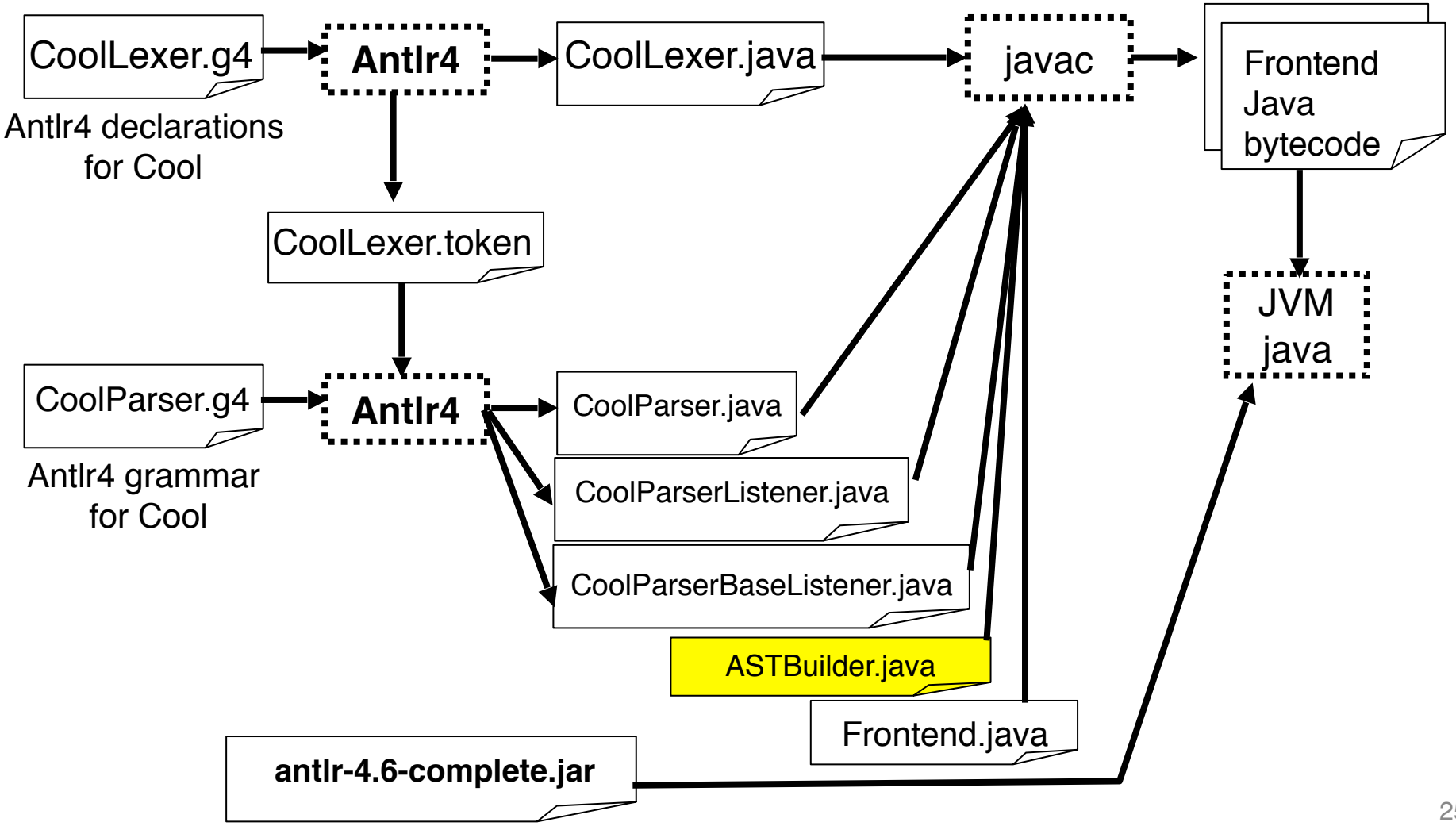
Antlr: build and run a parser



Antlr: build and run a parser



Antlr: build and run a parser



Antlr: build and run a parser