Presentation

Langage Grammar, AST, Eclipse Refactoring

arnaud.nauwynck@gmail.com

Plan

- Langage, Grammar, Compiler
- AST : Abstract Syntaxic Tree
  - Grammar to AST
  - Java AST
- Eclipse AST Support
  - Eclipse Overview for AST support
  - Custom Refactoring Actions

Langage Definition

Langage = file format supported by tool(s)

Compiler, Interpreter, Code Generator...

3 steps to define a langage

1) define Keywords, literal symbols

2) define Syntax

3) define Semantic

... 3 steps to process = scan, parse, process

Symbol Definitions (Lex)

Symbols =

- Keywords = « if », « for », ...
- Literal Values = 123.4, 'abc'
- Ignored tokens = comments, spaces, …

Scanner Implementation

- using regexps + states (Finite State Automaton)
  ex: float = [+-]?[0-9]*.[0-9]+(e[+-]?[0-9]+)?
- Tools: Lex (C), Flex, Jflex, …
- Javac (Jdk) scanner: hand-coded

Syntax Definition (BNF, Yacc)

Rules: <lhs> := <rhs1> <rhs2> .. <rhsN>

Rhs can be

- Terminal symbol (keyword, literal, …)
- Recursively defined by rule

Ex: arithmetic expr grammar

expr := value

expr := unaryop expr … op = -, !

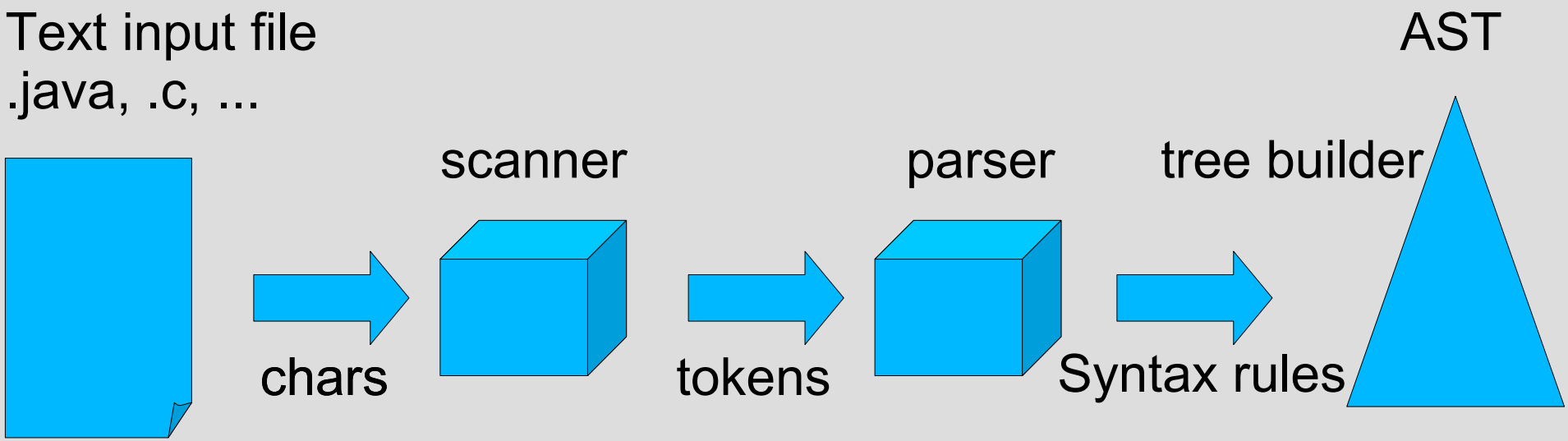
expr := expr binaryop expr … op= +, -, *, / ..

expr := '(' expr ')'

Parser Implementation

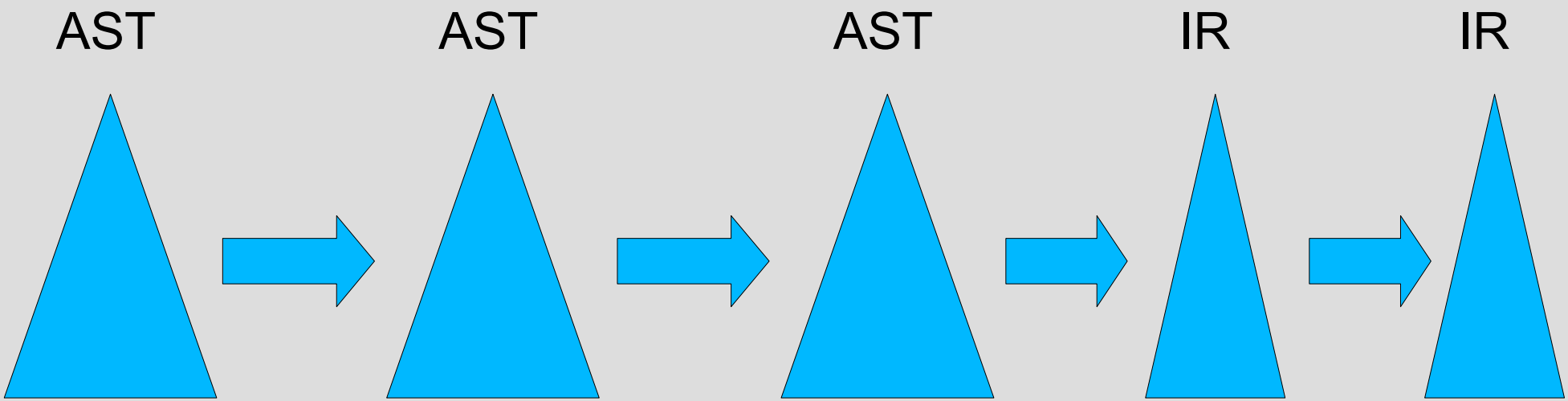
- Grammar types
  - LALR(1) : Look Ahead Left Recursive 1
  - LL(1)
- Compiler-Compiler Tools
  - Transform BNF rules to FSA
    (with shift-reduce algorithm)
  - Ex: Yacc (C), Javacc (Java), Jikes …
- Parsing Java:
  - In Jdk: LL hand-coded, in Eclipse: jikescc

Compiler Chain 1/3 : Frontend



AST = Abstract Syntaxic Tree
~ CST = Concrete Syntaxic Tree
(with spaces, comments,parenthesis ...)

Compiler Chain 2/3 : AAST (Attributed AST), IR



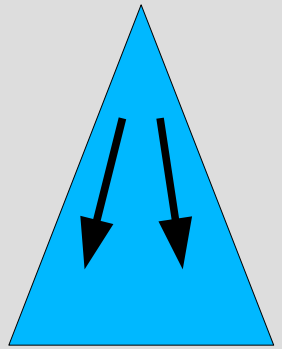
Transform
(change
tree structure)

Attribution
(add info to tree structure)
Ex: type resolution, symbol resolution,
type checking, ...

AST Attributes: Inherited, Synthetised, Mixed

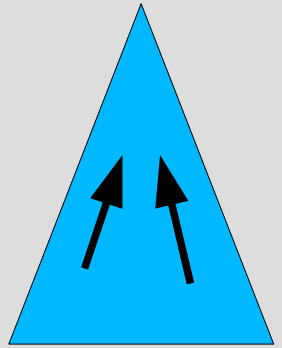
Ex of Inherited Attribute:

- Level in tree
- Context (list of ctx var decls)



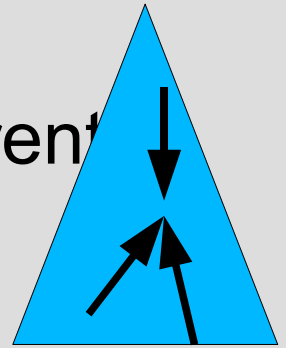
Ex of Synthethised Attribute:

- Depth of tree
- Declared Type



Ex of Mixed Attribute:

- Symbol resolution (lookup best symbol in parent ctx, knowing child type)



Tools for Attributed Grammars

Technologies for writing a compiler:

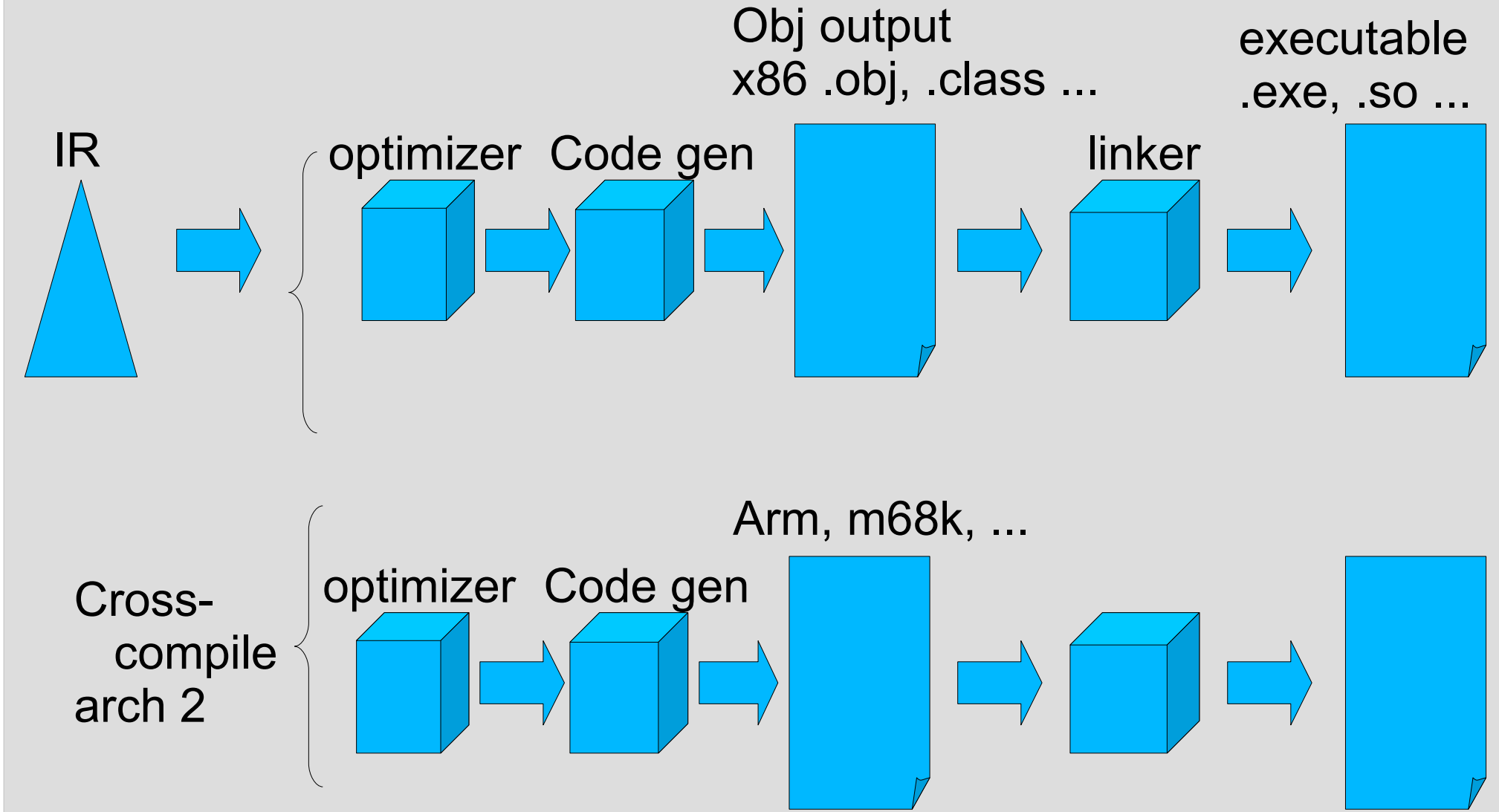
- Imperative Langage
  - < Fonctional Langage
    - < Attributed AST Specific Langage

Ex of tool implementation:

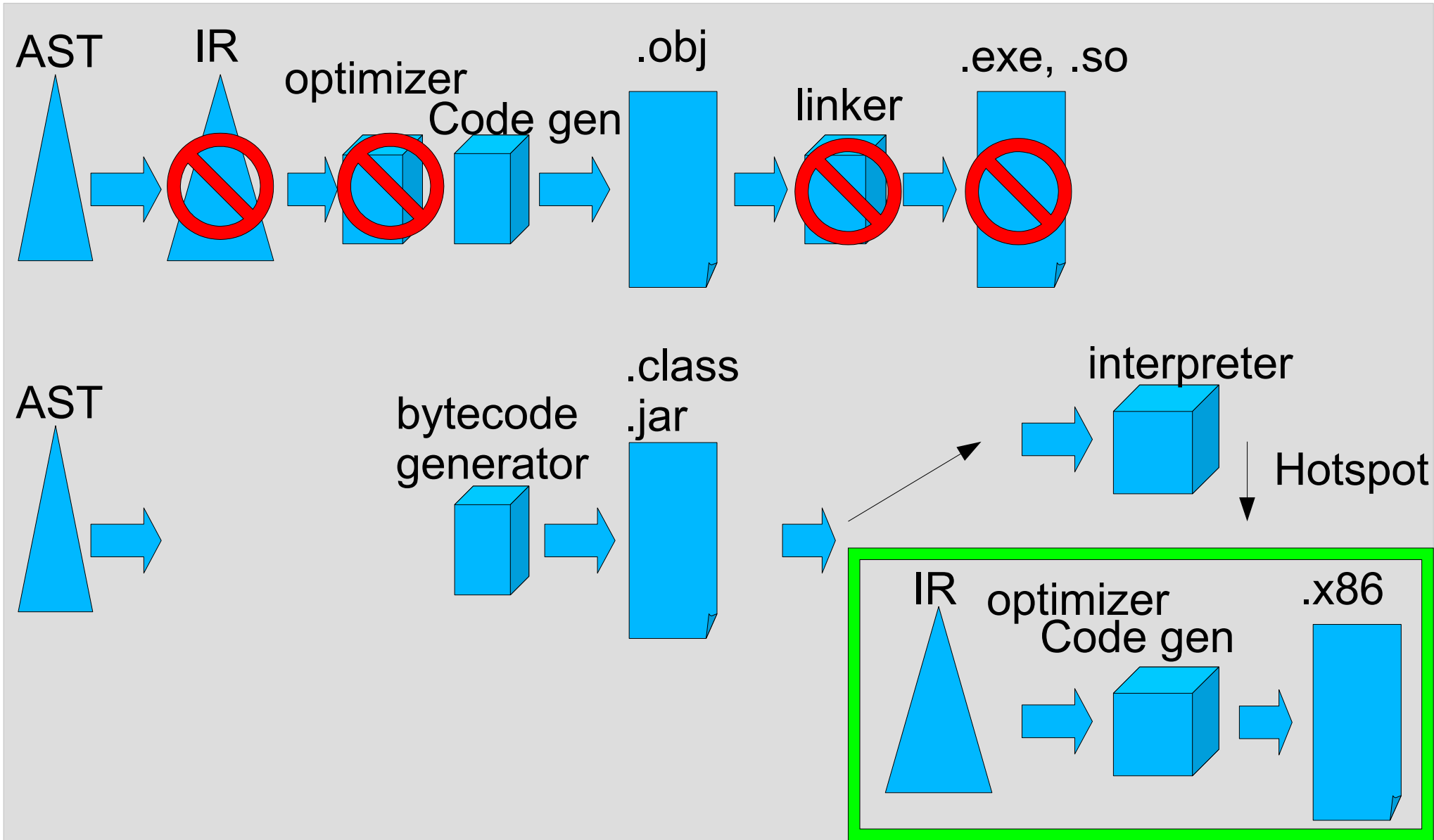
- Camel (fonctional langage... ex: Mlfi)
- OLGA (Inria … « Ouf, un Langage pour les Grammaire Attribuées »)

With Imperative form... need lot of unnatural code (Visitor design pattern, instanceof...)

Compiler Chain 3/3 : Backend



Compiler Chain in Jdk : Hotspot



Plan

- Langage, Grammar, Compiler
- AST : Abstract Syntaxic Tree
  - Grammar to AST
  - Java AST
- Eclipse AST Support
  - Eclipse Overview for AST support
  - Custom Refactoring Actions

Grammar => AST Class Hierarchy

1 BNF Rule = 1 structure (constructor)
=> 1 POJO Class

rules can share the same class
(ex: +-*/ => operator … for/do/while => loop)

Object-Oriented consistency : all classes
extends an AST root class

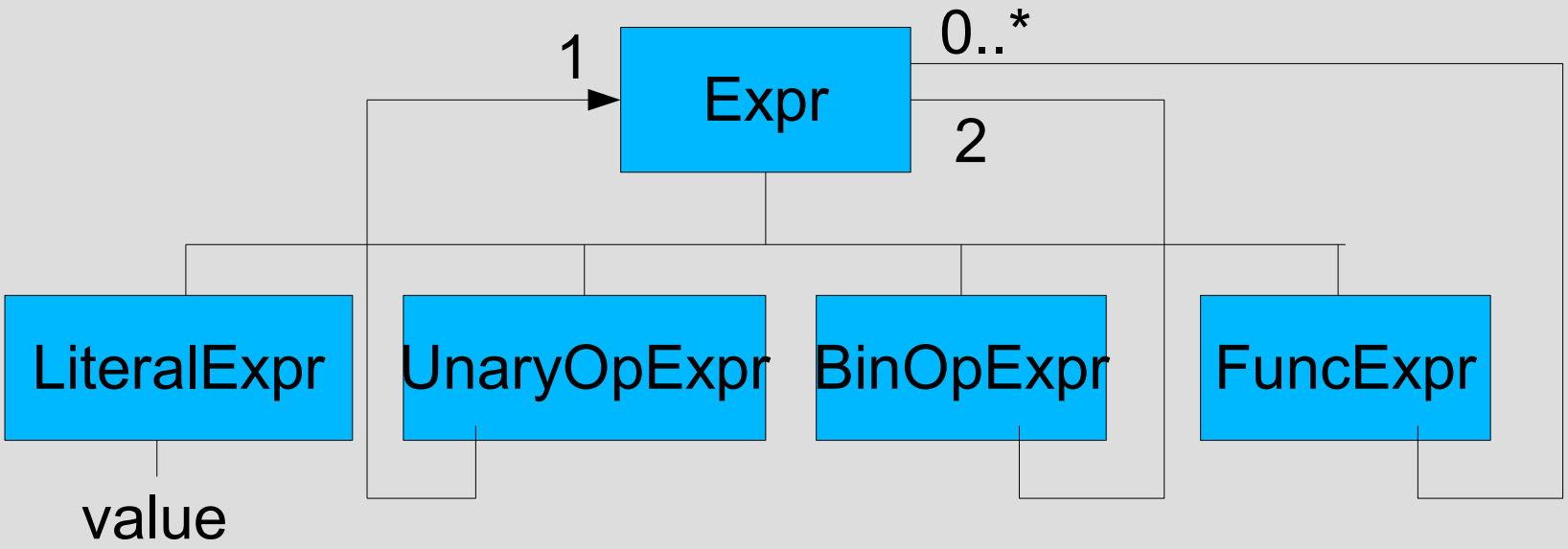
Ex: expr := expr '+' expr
{ return new BinaryOpExpr($1, $2, $3); }

AST Class Expr Sub-Hierarchy

expr := value

expr := unaryop expr … op = -, !

expr := expr binaryop expr … op= +, -, *, / ..



AST vs CST, Syntaxic Sugar

CST = Concrete Syntaxic Tree...
= contains structure
+ indent format + comments + style

AST = contains structure only

Ex for Expr:

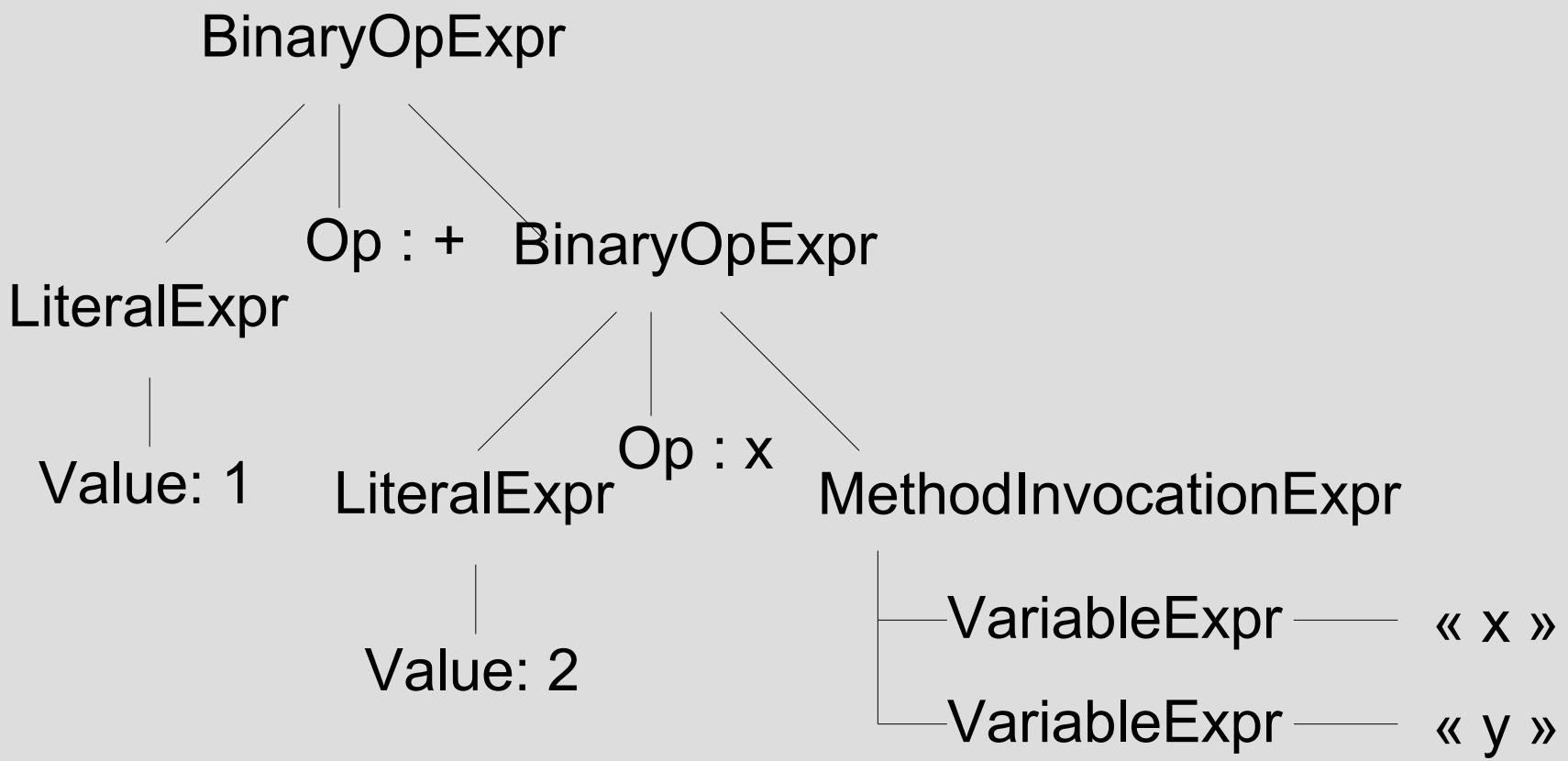
Expr := '(' expr ')' … not translated in AST!

Operator precedence

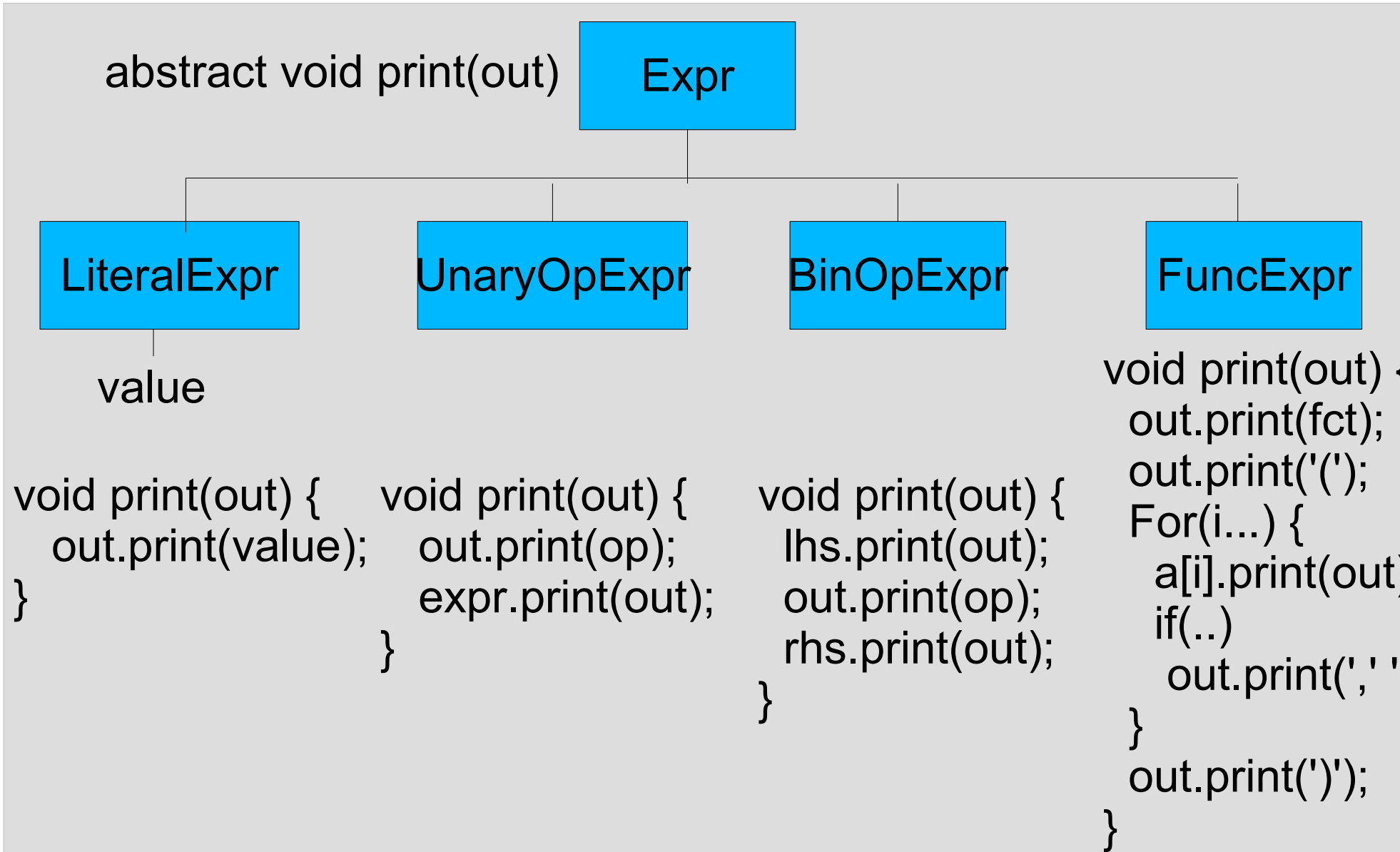
'expr + expr' could be written, '+(expr, expr)'

Sample AST Instance for Expr

Expr = 1 + 2 . f (x,y)



Sample AST Method: PrettyPrinter (AST back to Text)



abstract Methods => Visitor

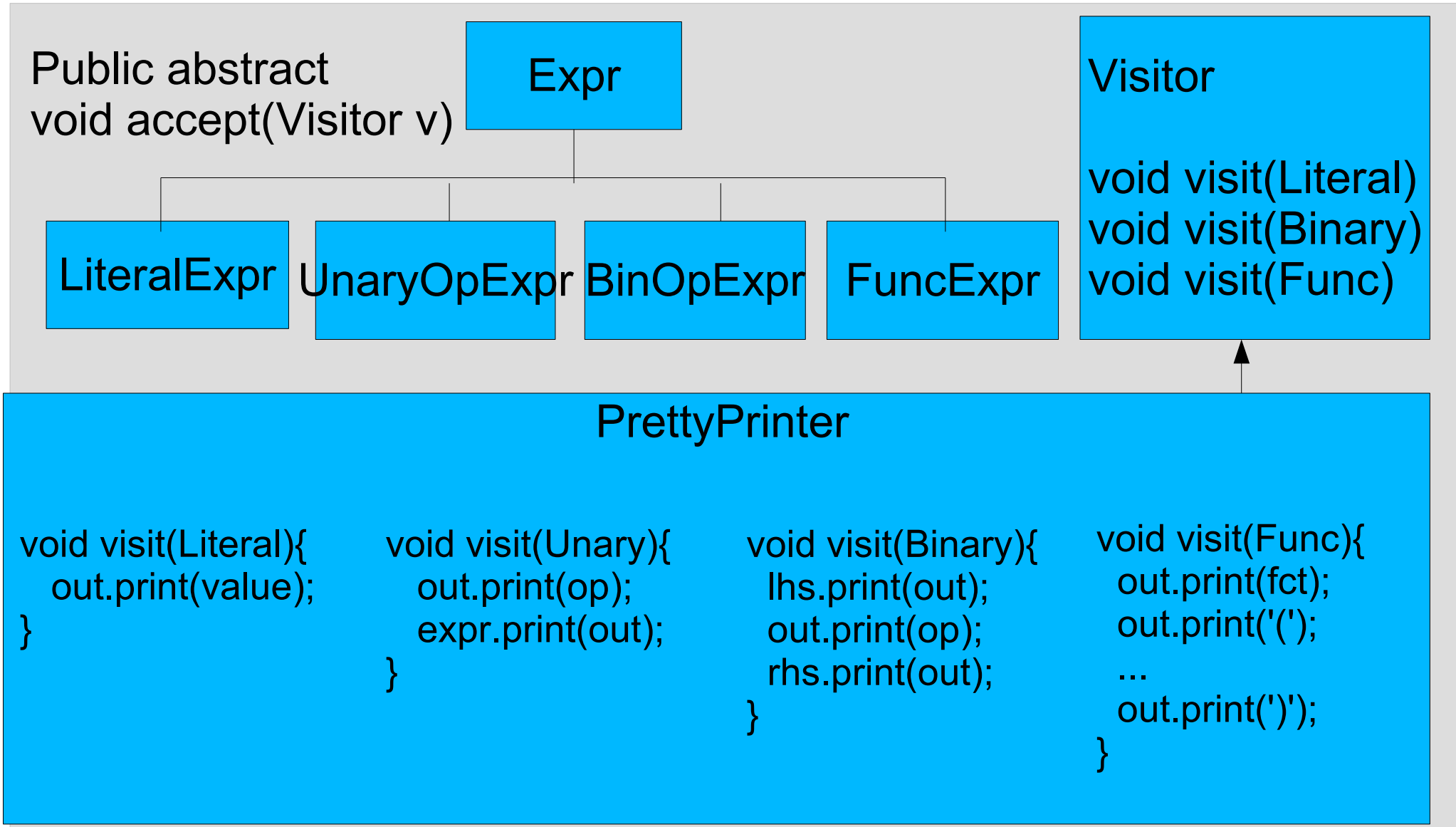
Visitor Design Pattern:

- Move code to separate files
- Keep AST classes as POJO
- Like a « switch-case » for object classes
- Name « visitor » for recursive traversing a tree

Implementation

- Abstract void accept(Visitor v);
- class X extends AST {
  void accept(Visitor v) { v.visitX(this); }

Visitor Design Pattern



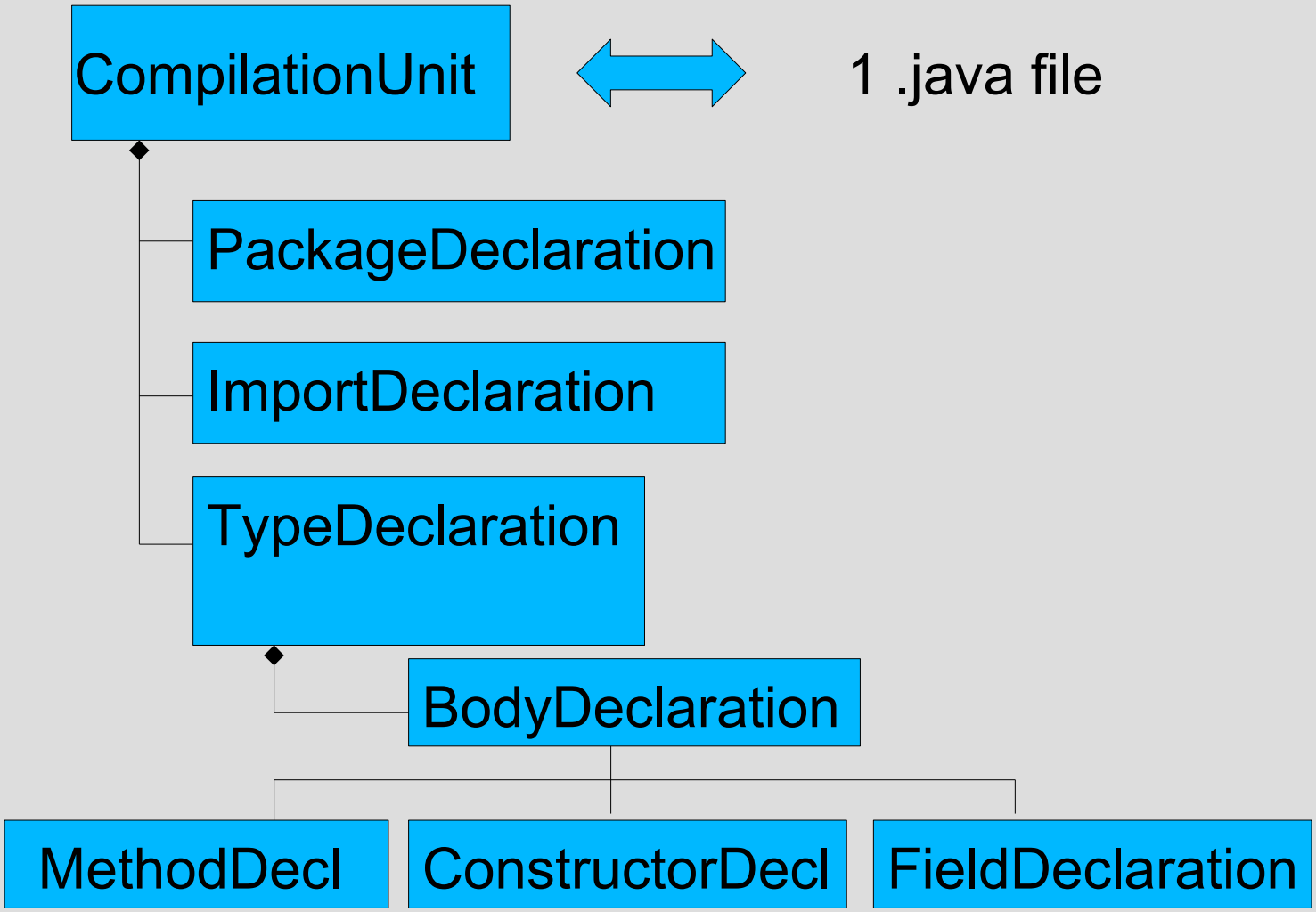
Parsing + AST for Java Langage

- Java = « simple » langage grammar
  - No preprocessor, No macros
  - Few Syntaxic sugar, verbose langage
  - Jdk5: Generics & Annotations
- Parsing Java : LL(1) grammar (recursive)
  - Already ~100 classes in AST
  - Do not reinvent the wheel!
  - Cf jsdk javac source … Tree + Pretty Printer
  - Cf Eclipse source … ASTNode

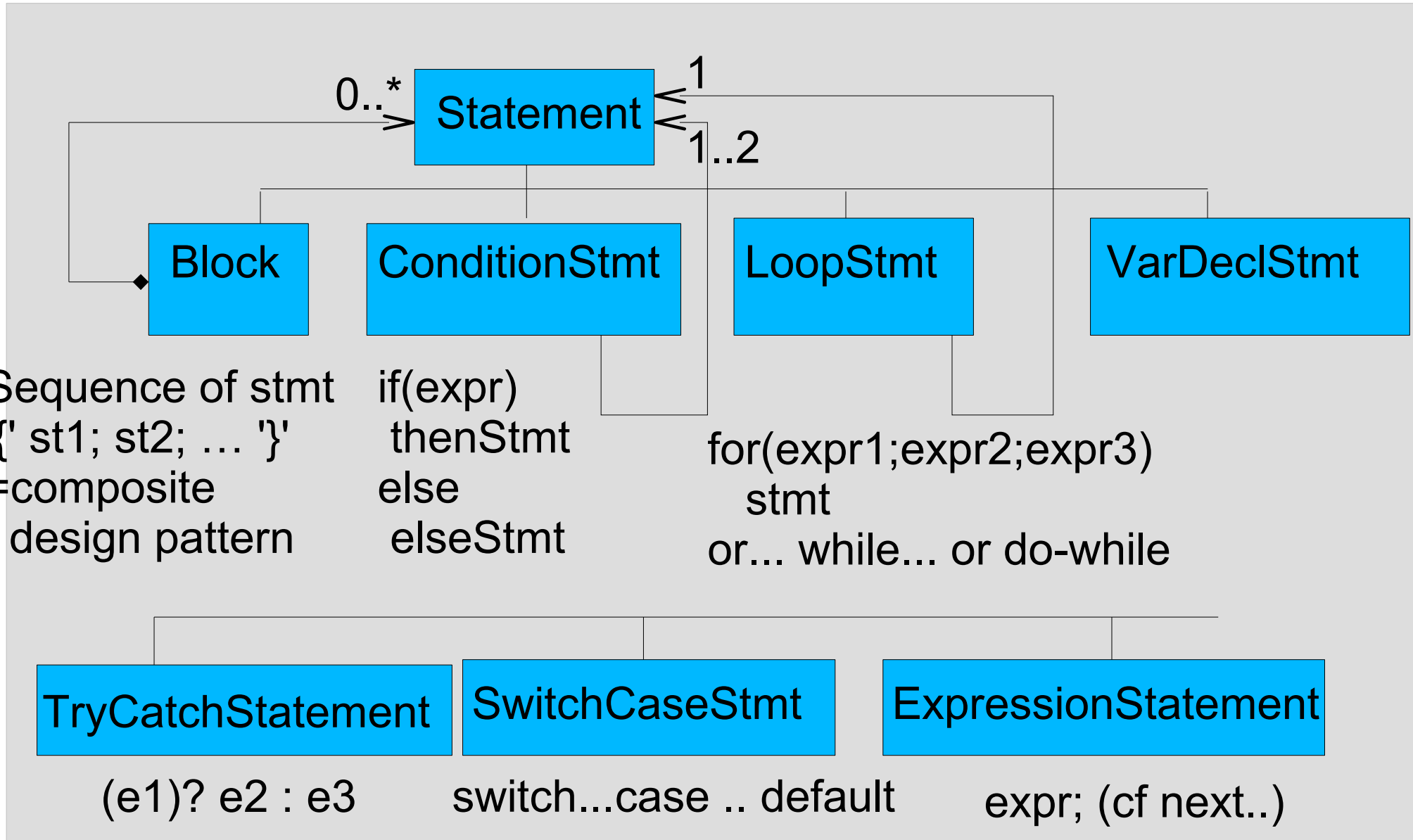
Java AST Explained

Declaration	Statement	Expression
Something with public/prot./private	Something with trailing ';' surroundable by '{ }'	Something surroundable by '( )'
has type (signature)	no type/value (= 'void')	has value / type
= Symbol	= Control Flow of imperative lang	= test & arith of lang
ex: Class, Interface, Method, Field	ex: If, Loop, Try-Catch	ex: +-*/, f(), ...

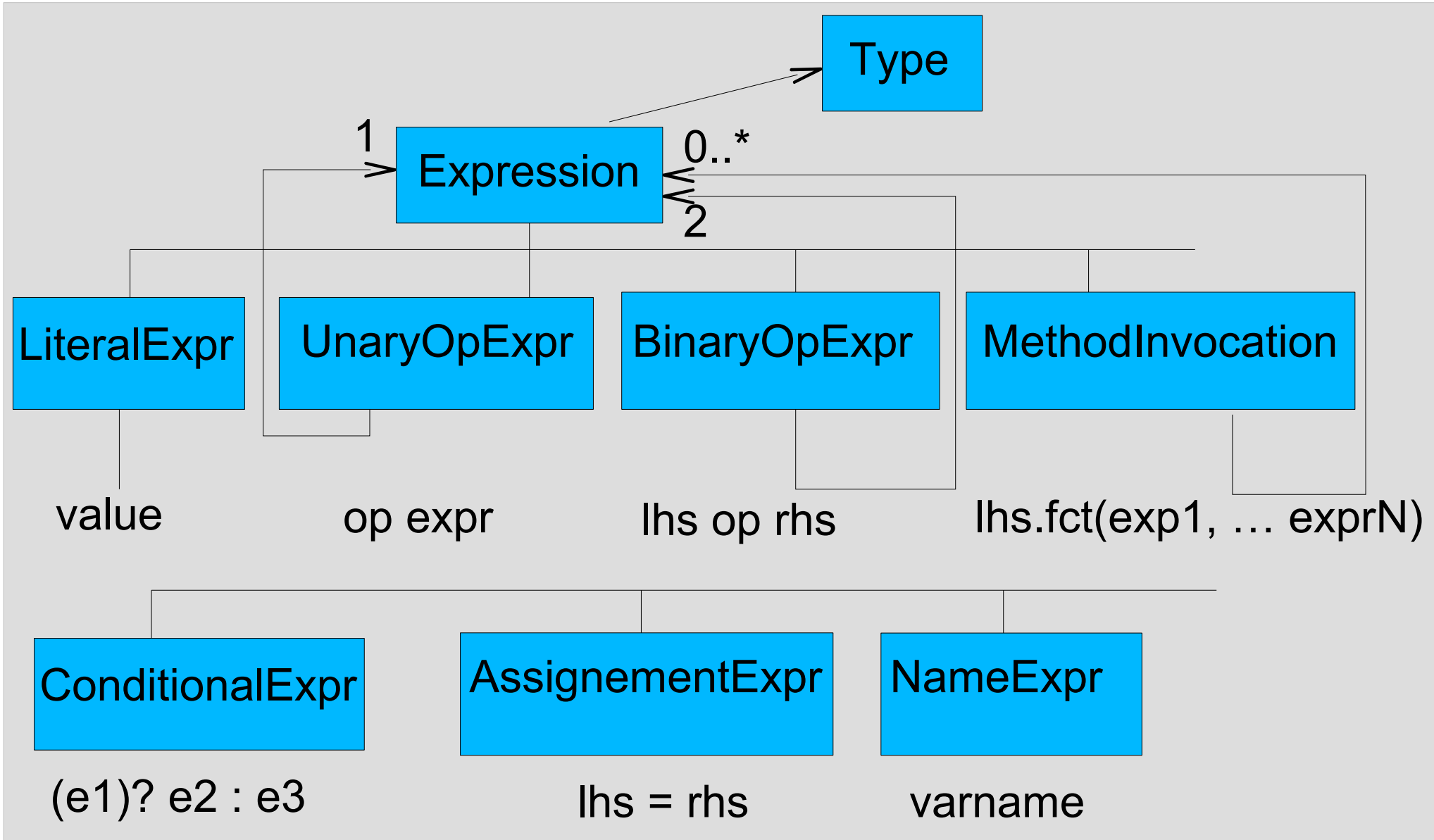
AST Declaration



AST Statement



AST Expression



ASTExpressionStatement

As name implies :

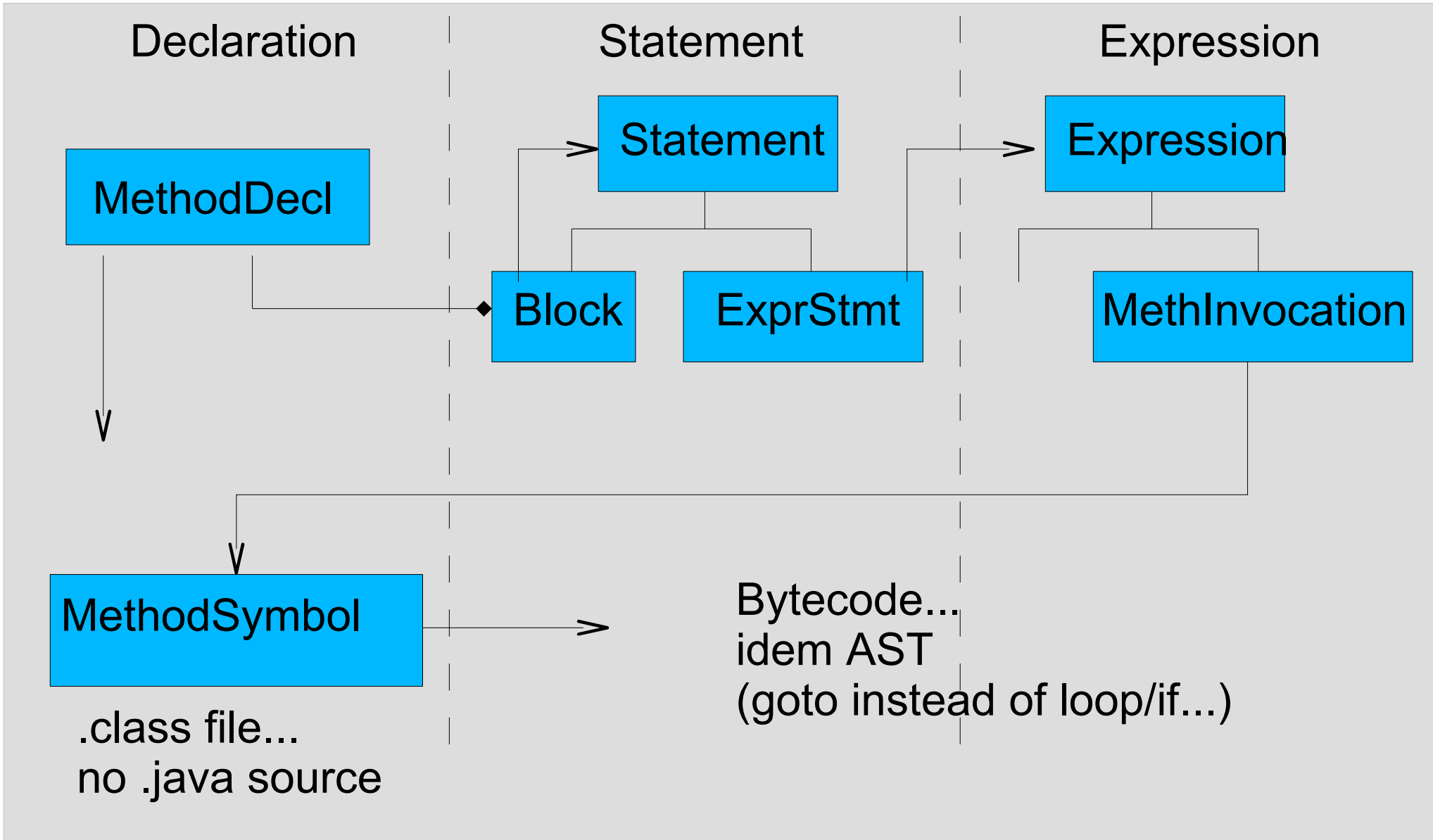
- It is a Statement
- Internally, is use/delegates to an Expression

Magic bridge for adding ';' to expr

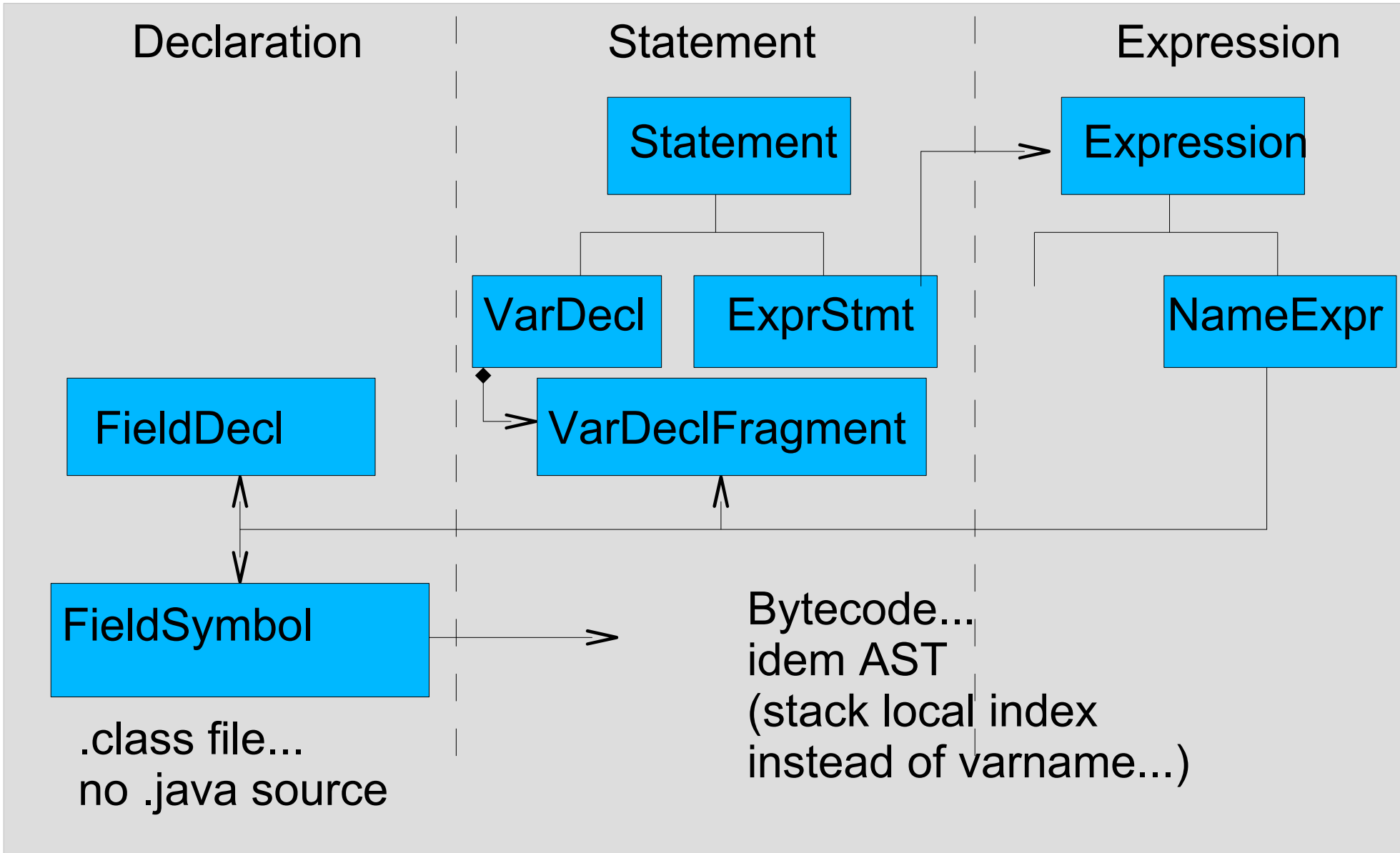
ex:

- fct(e1, ..eN);
  - = ExprStmt(MethodInvocation(..))
- lhs = rhs; … expr could be chained: a = b = c;
  - = ExprStmt(AssignExpr(lhs, rhs))

Declaration-Statement-Expression Method Body - Decl / Use



Declaration-Statement-Expression
Variable Decl / Use



Plan

- Langage, Grammar, Compiler
- AST : Abstract Syntaxic Tree
  - Grammar to AST
  - Java AST
- Eclipse AST Support
  - Eclipse Overview for AST support
  - Custom Refactoring Actions

Eclipse as Tool for AST

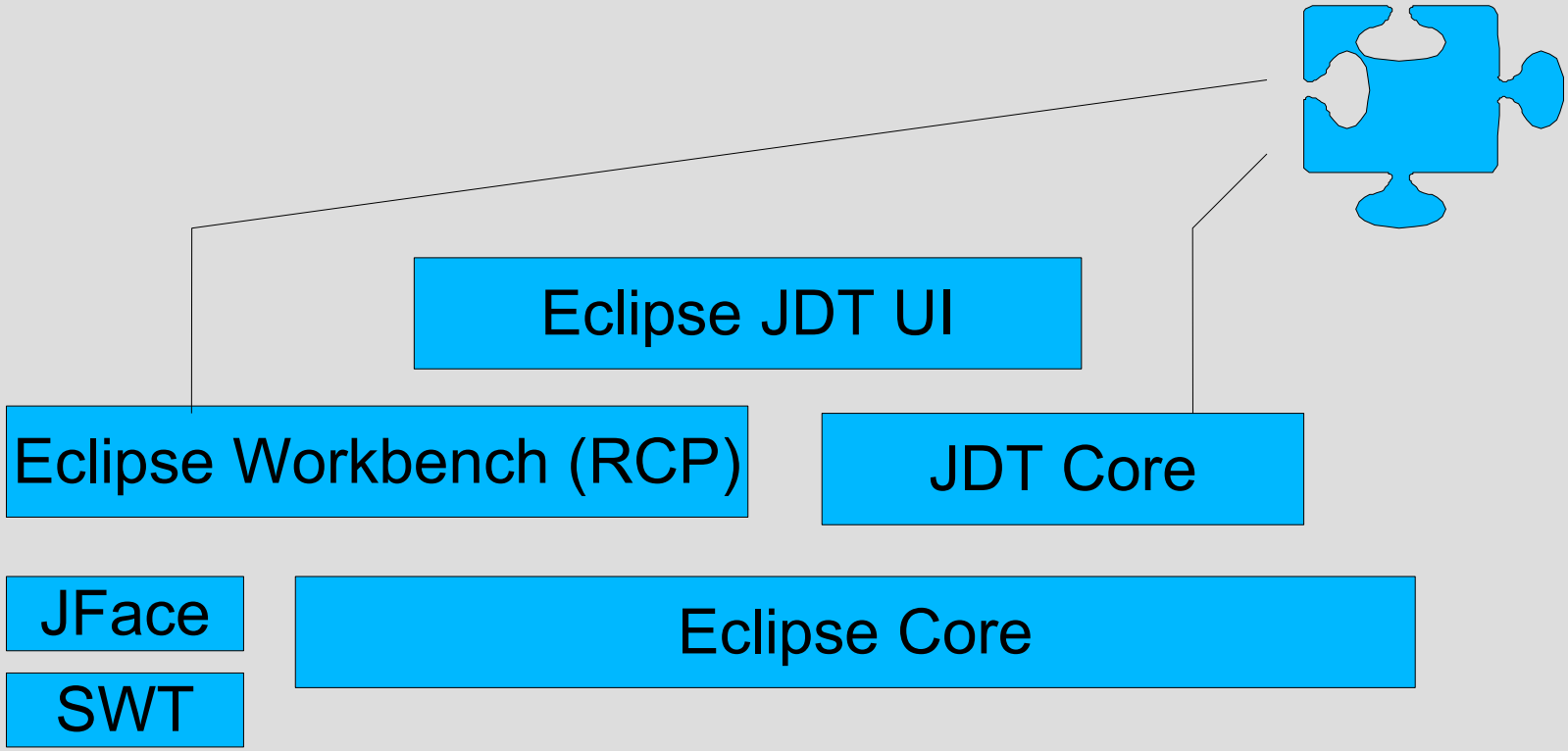
- Eclipse is Open-Source
- Highly extensible with plugins architecture
- High-level quality, very well designed
- Very Large community
- Rich support of Java
  - IDE with editor, views, navigators...
  - Compile (incremental)
  - AST is public API for Parse + read/write
  - Refactoring, search, advanced tools

Eclipse Plugin Architecture

To create your plugin...

Simply click « Create...> New Eclipse Plugin »

« Export...> Plugin Fragment » to dir dropins/



Adding Eclipse Context Menu

Edit file « plugin.xml »

Add objectContribution, menu, action...

Implement ActionDelegate class

Using Eclipse Internal Objects

3 hierarchies of objects... for distinct uses!

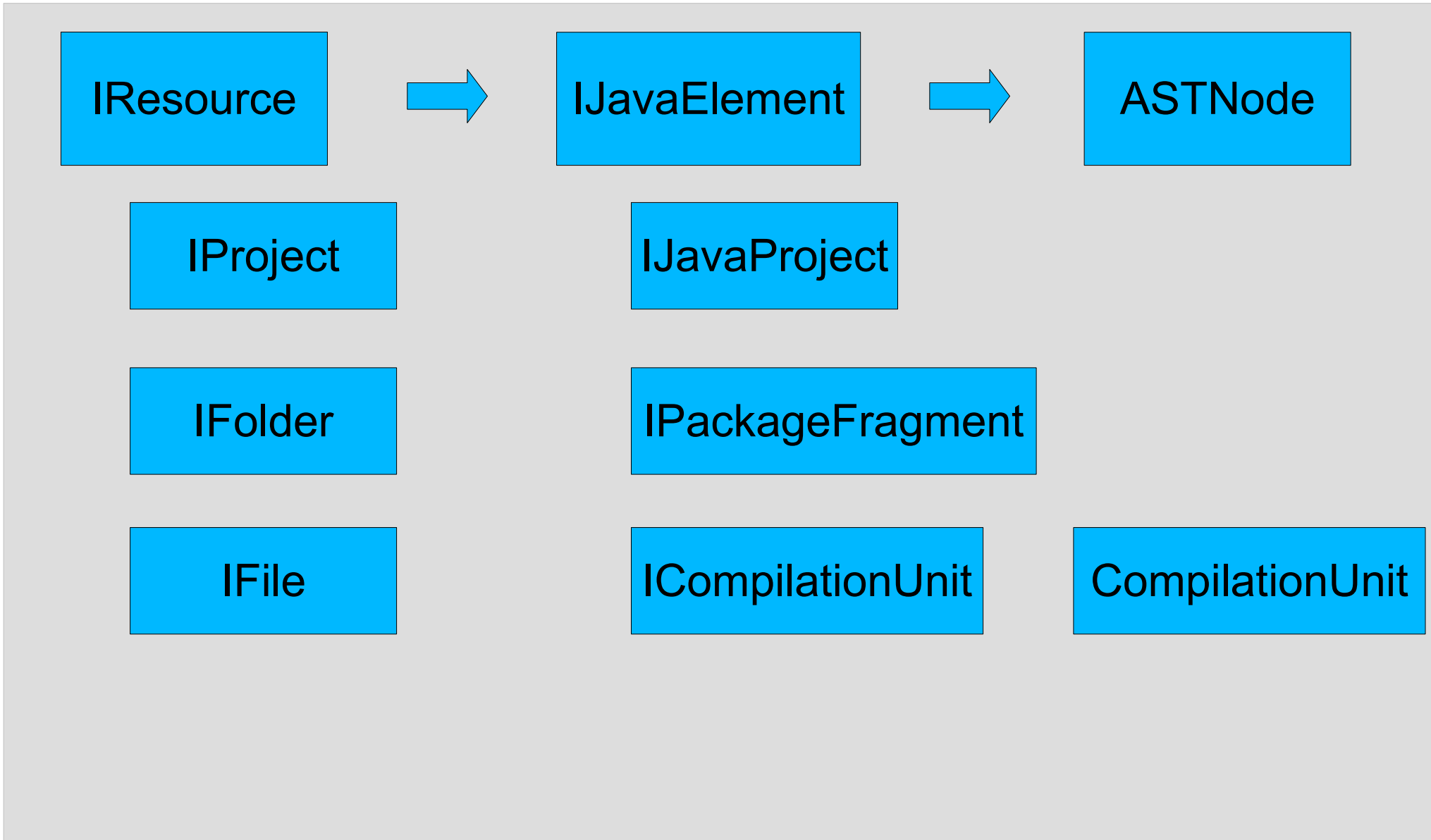
Convert current Selection to IResource(s),
then recurse on files => java => parse AST

Lightweight File wrapper
extension as VFS
root = « workspace »
mount = « project »
facade for all FS ops

Lightweight Java objects
(fast repository
for java navigation)

Heavy AST...
(1Mo per file)
for tmp work
read/write

Eclipse Internal Objects(2)



Plugin Summary: Action-Selection-IResource-Java-AST

Sample Outputs for AST

Plugins Example:

- Code Analyzer
- Extended Tree-Regexp Search

Output:

- Write to file
- Copy text to clipboard
- Text console view (~ 20 lines with ConsolePlugin)
- Create an Eclipse Table View...

Sample Code Analysis Plugin

Project Meta Rule : every try-catch should
rethrow ex
...or call « ExUtil.ignore() »

Code:

```
New ASTVisitor() {
   public boolean visit(TryStmt p) {
     Block catchBlock = p.get..();
     Statement last = ...;
     if (last …)
```

Refactoring = AST in Write mode

Step 1: declare
compilationUnit.recordModifications()

Step 2: call ASTNode setters

node.delete();

node.set...();

nodeList.add(..) / remove(..)

Remark: Tree fully navigable
node.getParent() … node.getChildLocation()

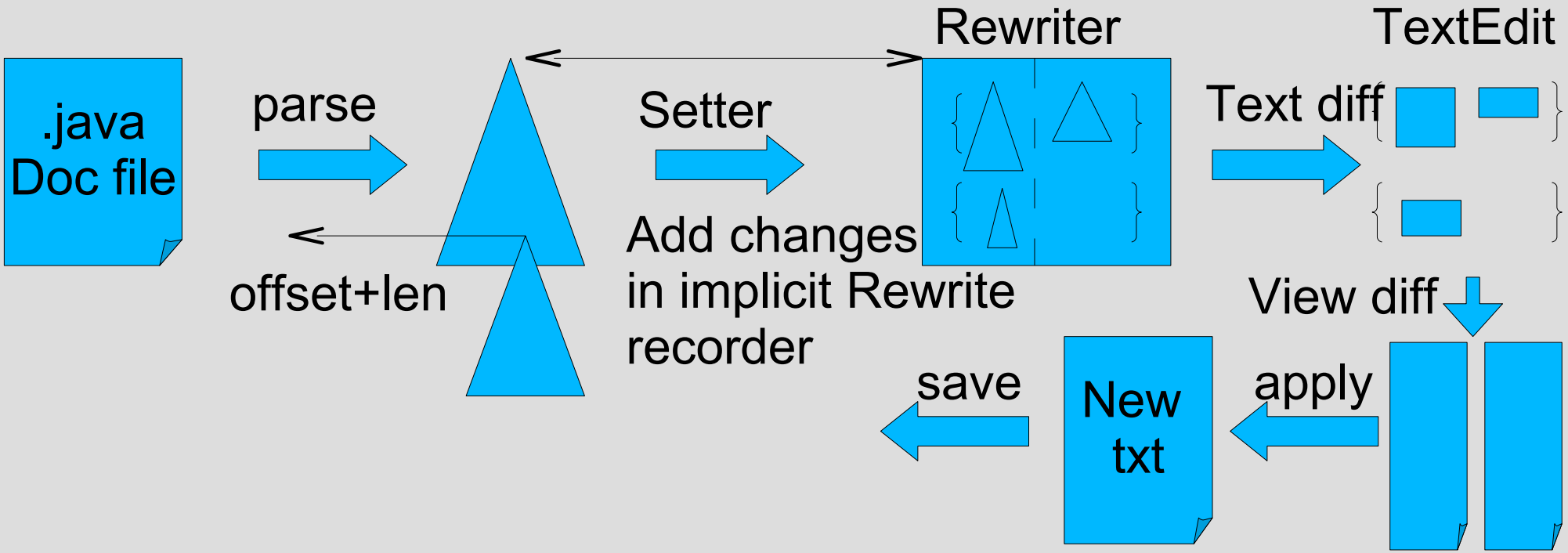
Step 3: display/flush text file changes

AST recordModifications … implicit Rewriter + TextEdit

AST Diff (insert/update/delete) are recorded

AstNode knows its char offset + len

=> ast diff converted to text diff



Refactoring Business Code

```
Class … extends MyRefactoringHelper {
    public Object prepareRefactoring(
                                CompilationUnit cu) {
        … my detector + memento
        return memento;
    }
  public void doRefactor(CompilationUnit cu,
                                Object memento) {
        … do modify cu  using memento
    }
```

Refactoring Boilerplate Code

```
Abstract class MyRefactoringHelper {
    public void execute(
      Collection<ICompilationUnit> icus) {
      for(ICompilationUnit icu : icu) {
        CompilationUnit cu = … parse(icu);
        Object m = prepareRefactoring(cu);
        if (m != null) {
          cu.recordModifications();
          TextEdit te = cu.rewrite();
          Document doc = …
          applyTextEdit(doc, te);
         DocumentManager.commit(doc); }  }  }
```

Sample Jdk5 Custom Refactoring

Code before:

```
Collection ls = ...
for(Iterator it = ls.iterator(); it.hasNext(); ) {
    XX elt = (XX) it.next();
```

knowing jdk5 implementation of generic...
Code is equivalent to

```
Collection<XX> ls = ...
for(XX elt : ls) { … }  // may throw ClassCastEx !
```

Jdk5 Generic Refactoring Steps

Intermediate Step1:

```
Collection<XX> ls = ...
   for(Iterator<XX> it=ls.iterator(); it.hasNext();) {
       XX elt =    it.next();
```

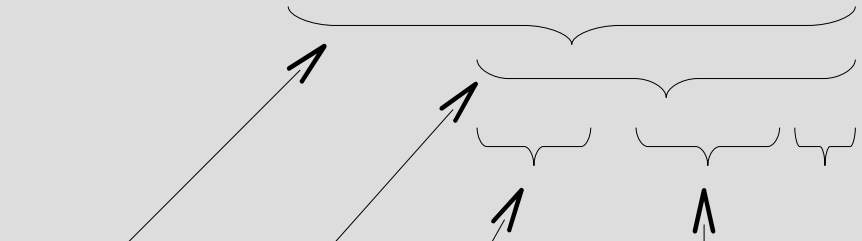
add generic type info
remove useless downcast

...Step 2: apply existing « cleanup action » :
select « code style > use foreach loop »

Generic Iterator Detection

Pattern 1 to detect:

(XX) iter.next()



Corresponding AST Pattern:

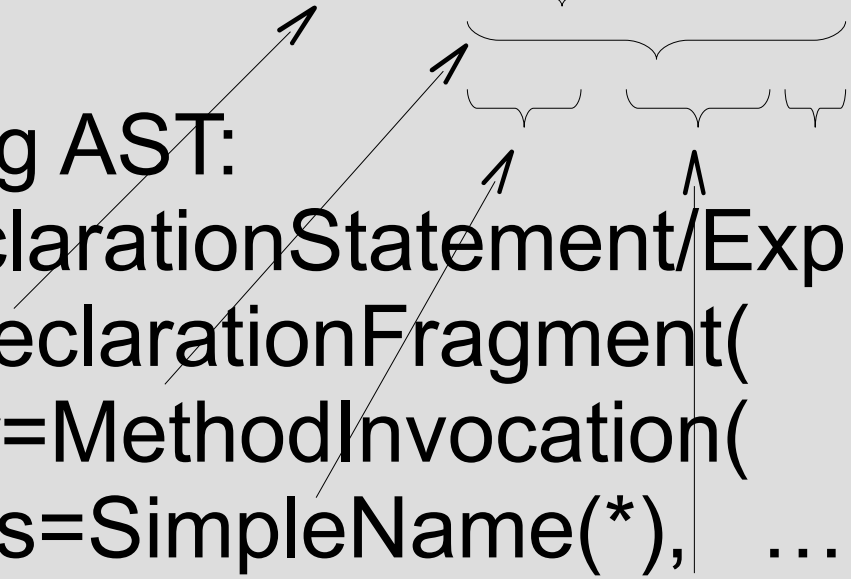
```
CastExpr(
    MethodInvocation(
        lhs=SimpleName(*),
        methodName=« next »,
        args=emptyList))
```

Iterator Declaration Detection

Pattern 2 to detect:

variable « iter » must be declared above as

Iterator iter = ls.iterator()



Corresponding AST:

```
VariableDeclarationStatement/Expr(
  VariableDeclarationFragment(
    initExpr=MethodInvocation(
        lhs=SimpleName(*),   … optional
        methodName= « iterator »,
        args={} )))
```

Adding <Type> to Iterator/Coll

Pattern 1

Before: Iterator it = … after: Iterator<XX> it = ..

Optional pattern 2

Before: Collection ls... after: Collection<XX> ls

AST rewrite:

Replace type=SimpleName(name)

By type= GenericType(name,
list(SimpleName(« XX »)))

Enhancing Custom Jdk5 Refactoring

Several defaults for this naive refactoring

Detect iterator var defined outside of loop

Detect while() loop

Too few effects... Should also detect list.add() / list.remove()

Should propagate signature changes (break interface/impl @Override link)

Merge with existing eclipse action!
« Refactor... > Infer generic type... »

More Advanced Refactoring Analysis

Propagating Type in use-def Inter/Intra procedure call graph in NOT SO EASY...

Intra-procedure «Graph Solving Fwk» is NOT in standard eclipse!

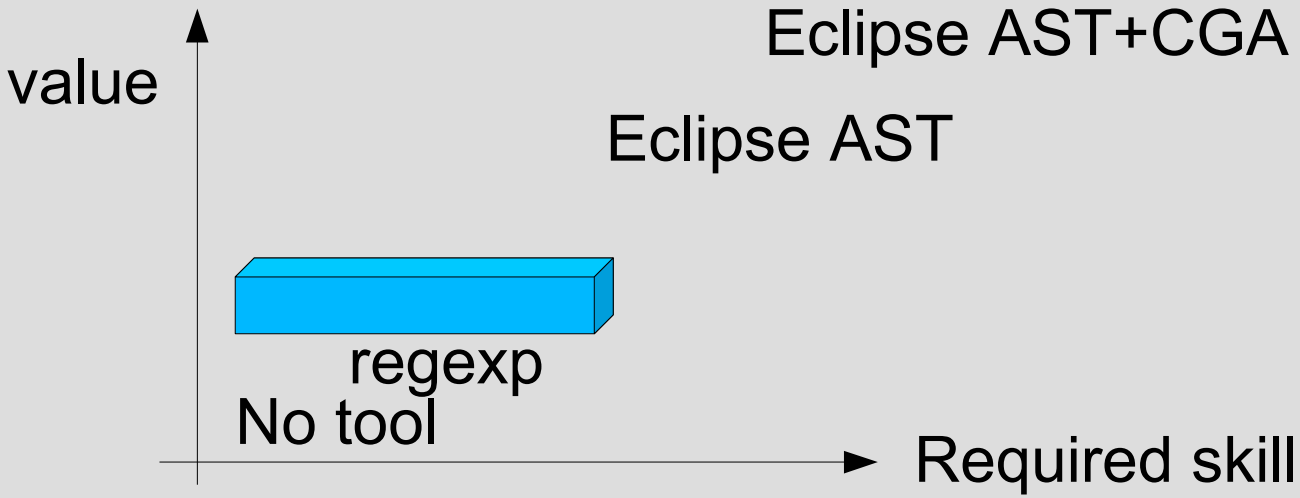
See for example

- Sablecc fwk (not in eclipse)
- Eclipse wala project (sourceforge, formely Ibm Watson research project)

Conclusion

Writing your own Eclipse Refactoring plugins

- Is possible & very very powerfull
- Is difficult only at first seeght
- Use as script langage for prototype/1 time usage
- Use to adapt your project meta architecture



Questions

arnaud.nauwynck@gmail.com