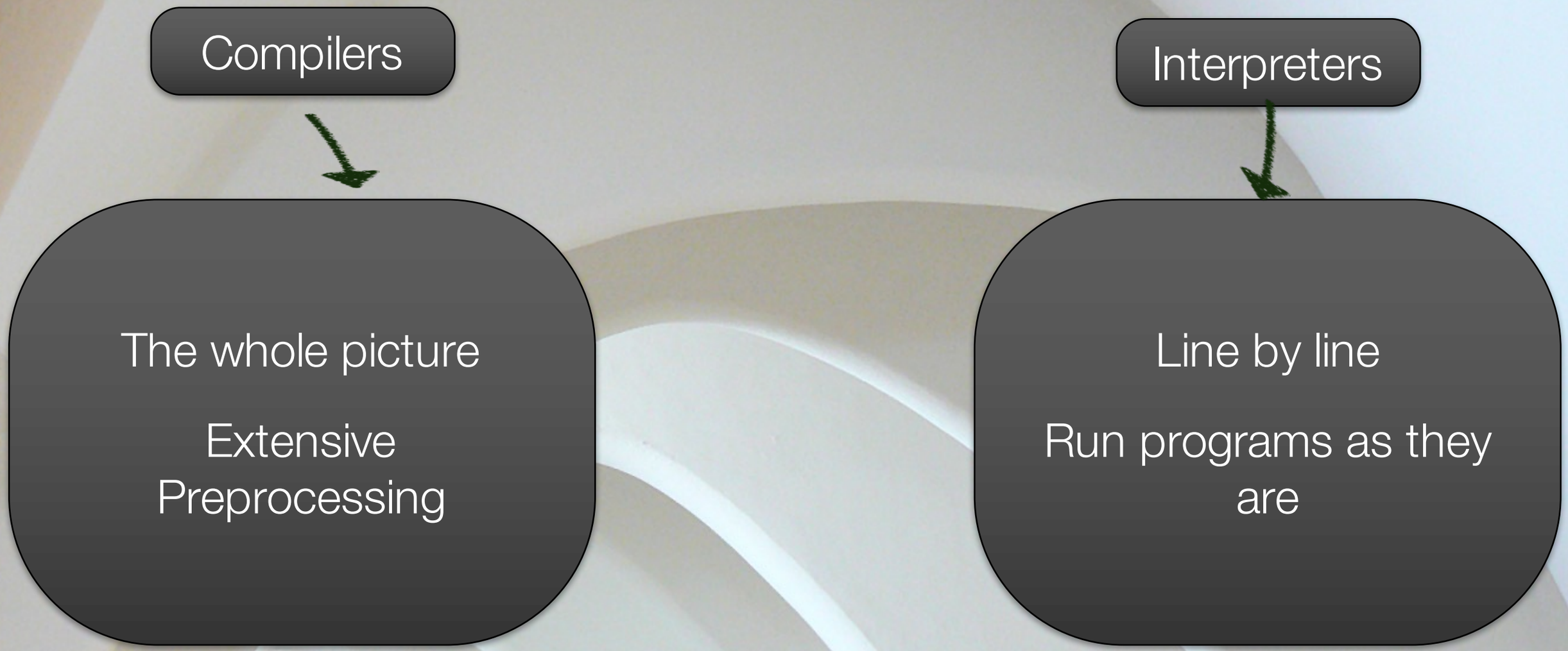
Java Compilers

2016

Outline

- Introduction
- History
- Compiler Structure
- Jflex and Cup
- Jasmine
- References

Introduction



Where they came from?

1954: IBM Develops the 704 – Assembly code

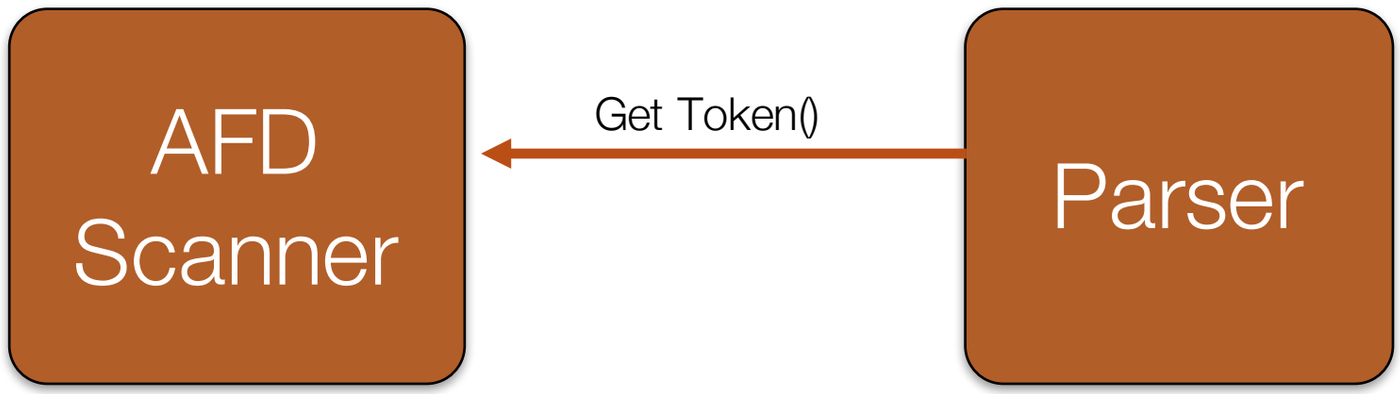
SpeedCoding: interpreter (10-20 times slower)

Fortran I: John Backus

- High Level Code to Assembly
- 1958: 50% software is in Fortran!
- First Compiler!

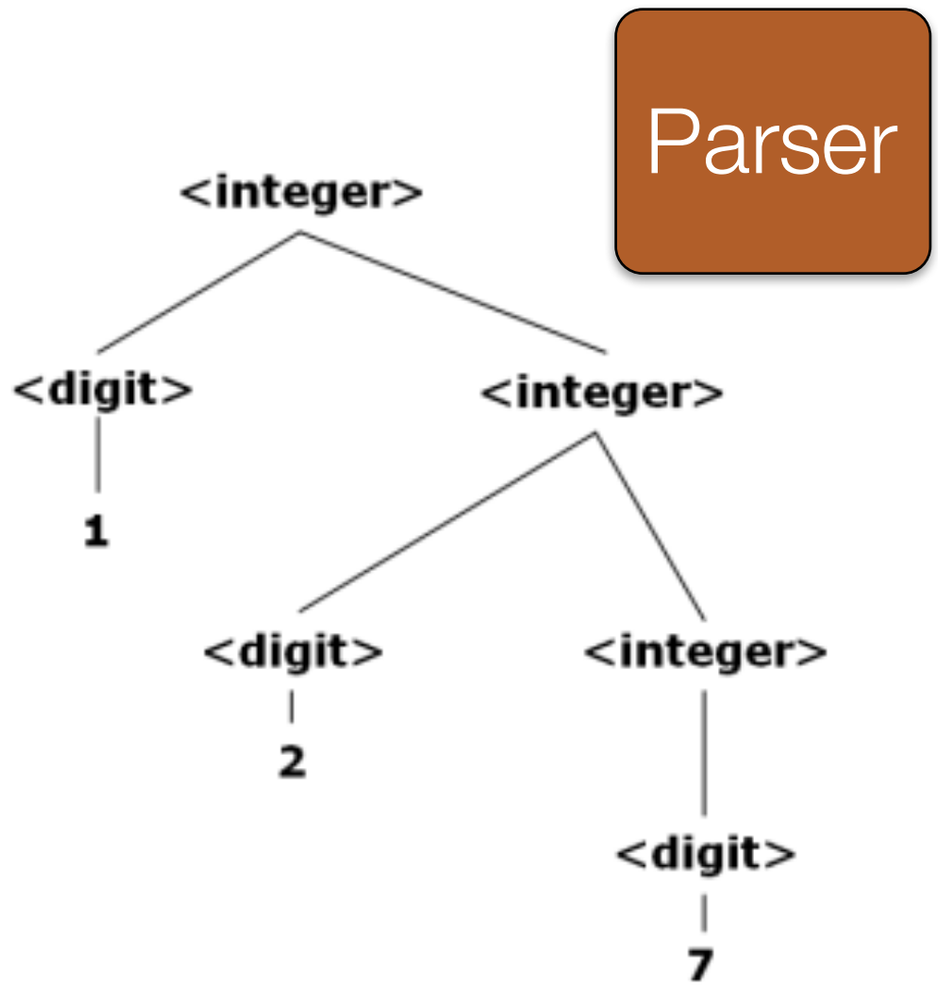
Compiler Structure

1. **Lexical Analysis**
2. **Parsing**
3. Semantic Analysis
4. Optimization
5. Code Generation



Compiler Structure

1. Lexical Analysis
2. Parsing
3. **Semantic Analysis**
4. Optimization
5. Code Generation

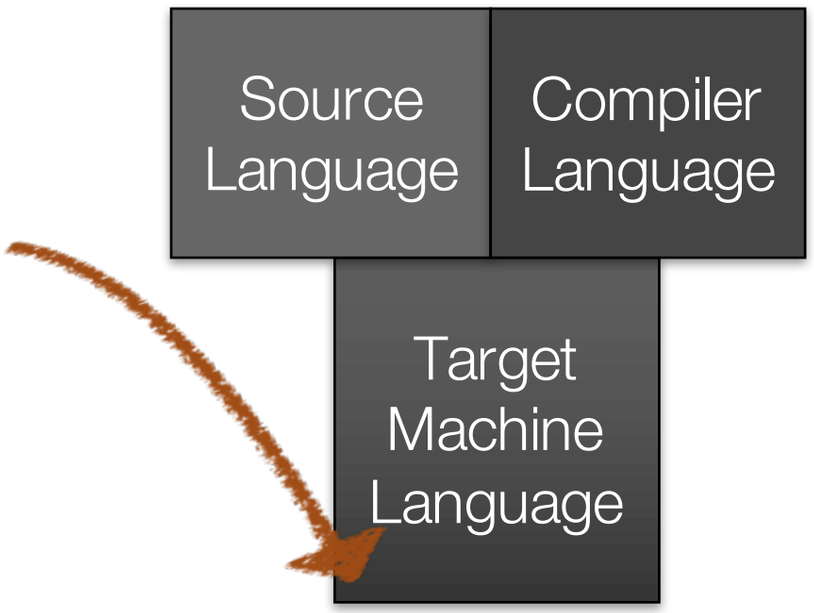


Compiler Structure

1. Lexical Analysis
2. Parsing
3. Semantic Analysis
4. **Optimization**
5. Code Generation

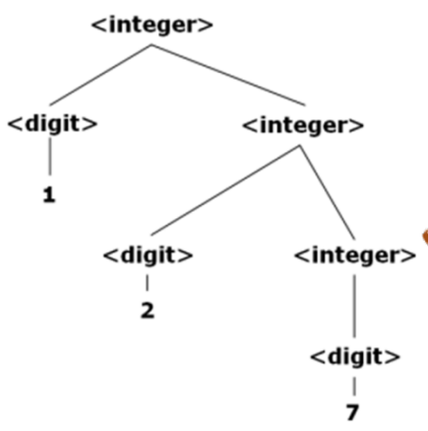
```
int X = 3 + 7;
```

```
int X = 10;
```



Compiler Structure

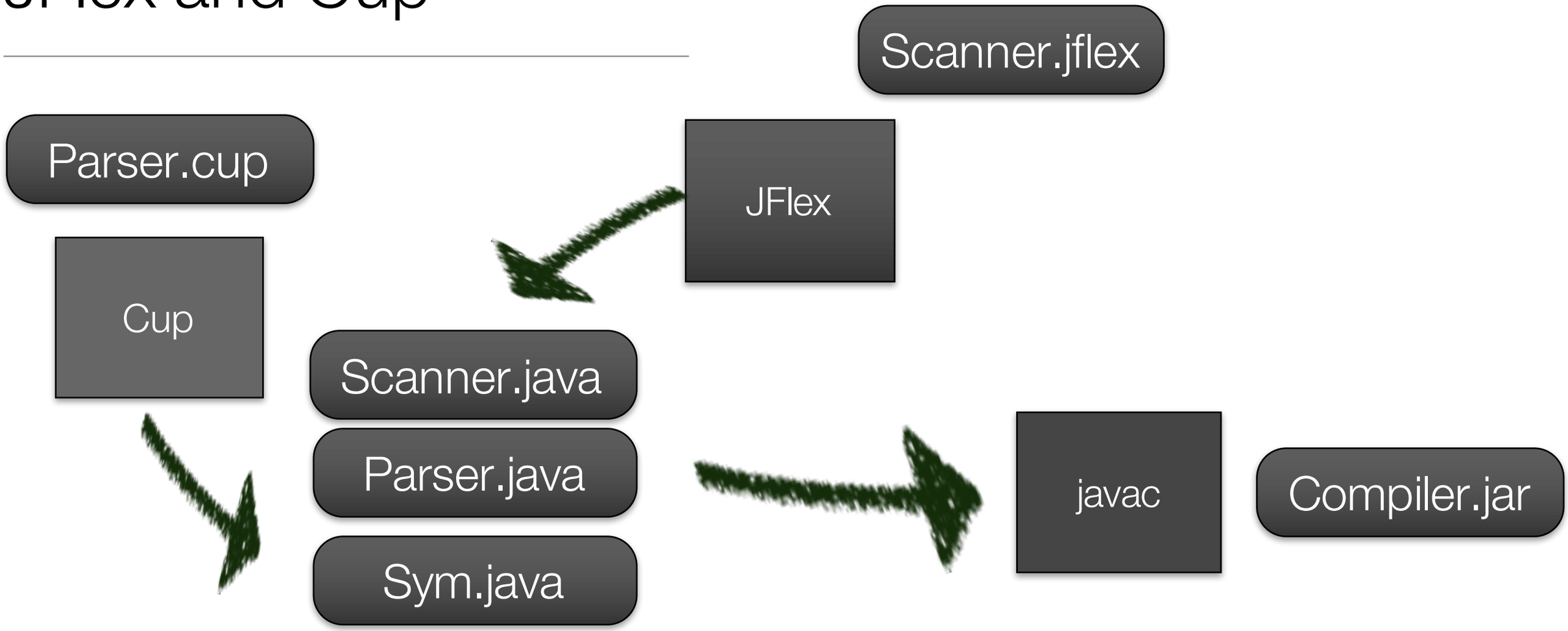
1. Lexical Analysis
2. Parsing
3. Semantic Analysis
4. Optimization
5. **Code Generation**



IL: Intermediate Code

```
0400 2073FE JSR $FE73        s~
0403 A200   LDX #$0          "□
0405 BD8004 LDA $480,X       =□␁
0408 F006   BEQ $410         p✓
040A 2075FE JSR $FE75         u~
040D E8     INX              h
040E D0F5   BNE $405         Pu
0410 00     BRK              □
0411 B9     *=$480
0480 48     'H               H
0481 45     'E               E
0482 4C     'L               L
0483 4C     'L               L
0484 4F     'O               O
0485 00     $0               □
0486 67     !
```

JFlex and Cup



Parser.cup

```
 package Example;

import java_cup.runtime.*;

parser code {:
        public static void main(String args[]) throws Exception {
                SymbolFactory sf = new DefaultSymbolFactory();
                if (args.length==0) new Parser(new Scanner(System.in,sf),sf).parse();
                else new Parser(new Scanner(new java.io.FileInputStream(args[0]),sf),sf).parse();
        }
:}

terminal SEMI, PLUS, TIMES, LPAREN, RPAREN;
terminal Integer NUMBER;

non terminal expr_list, expr_part;
non terminal Integer expr;

precedence left PLUS;
precedence left TIMES;

expr_list ::= expr_list expr_part | expr_part;
expr_part ::= expr:e {: System.out.println(" = "+e+";"); :} SEMI;
expr      ::= NUMBER:n
                {: RESULT=n; :}
            | expr:l PLUS expr:r
                {: RESULT=new Integer(l.intValue() + r.intValue()); :}
              | expr:l TIMES expr:r
                {: RESULT=new Integer(l.intValue() * r.intValue()); :}
              | LPAREN expr:e RPAREN
                {: RESULT=e; :}
```

Scanner.jflex

```
 package Example;

import java_cup.runtime.SymbolFactory;
%%
%cup
%class Scanner
%{
        public Scanner(java.io.InputStream r, SymbolFactory sf){
                this(r);
                this.sf=sf;
        }
        private SymbolFactory sf;
%}
%eofval{
   return sf.newSymbol("EOF",sym.EOF);
%eofval}

%%
";" { return sf.newSymbol("Semicolon",sym.SEMI); }
"+" { return sf.newSymbol("Plus",sym.PLUS); }
"*" { return sf.newSymbol("Times",sym.TIMES); }
"(" { return sf.newSymbol("Left Bracket",sym.LPAREN); }
")" { return sf.newSymbol("Right Bracket",sym.RPAREN); }
[0-9]+ { return sf.newSymbol("Integral Number",sym.NUMBER, new Integer(yytext())); }
[ \t\r\n\f] { /* ignore white space. */ }
. { System.err.println("Illegal character: "+yytext()); }
```

Jasmine

```
.class public HelloWorld.j
  .super java/lang/Object

  .method public <init>()V
     aload_0
     invokenonvirtual java/lang/Object/<init>()V
     return
  .end method

  .method public static main([Ljava/lang/String;)V
     .limit stack 2
     .limit locals 2
     getstatic      java/lang/System/out Ljava/io/PrintStream;
     ldc            "Hello World."
     invokevirtual  java/io/PrintStream/println(Ljava/lang/String;)V
     return
  .end method
```

Bytecode

```
;
;;Classfile version:
;    Major: 45
;    Minor: 3

.source mymath.java
.class  public synchronized mymath
.super  java/lang/Object

; >> METHOD 1 <<
.method public <init>()V
    .limit stack 1
    .limit locals 1
.line 1
    aload_0
    invokenonvirtual java/lang/Object/<init>()V
    return
.end method

; >> METHOD 2 <<
.method public static sum(II)I          ; takes two int args and returns an int result
    .limit stack 2
    .limit locals 2
.line 3
    iload_0                      ; load integer arguments (passed as local
    iload_1                      ;   variables 0 and 1) onto stack
    iadd                        ; integer add, leaving result on stack
    ireturn                     ; return integer on top of stack
.end method
```

```
public class mymath {
  public static int sum (int a, int b) {
    return a + b;
  }
}
```

References

- Dragon Book (Aho, Sethi, Ullman)
- Compiladores, Teoría e Implementación (Jacinto Catalán)
- http://www.tldp.org/LDP/LGNET/issue41/sevenich.html
- Programming Languages Pragmatic, Scott, 2009
- Practica Foundations for Programming Languages, Harper,2