Jay Lexical Specification

Identifiers	
Integer literals	$0\|[1-9][0-9]*$
Boolean literals	`true false`
Separators	`( ) { } ; ,`
Operators	`= > < ! == <= >= != || && + - * /`
Keywords	`public class static String[] args void main System.out.println boolean else if int while`

Jay Concrete Syntax

Program	→	`public class` ID '{' `public static void`
		`main ( String[] args )`
		'{' *Declarations Statements* '}' '}'
Declarations	→	*Declartion* ∗
Declaration	→	*Type Identifiers* ;
Type	→	`int` | `boolean`
Identifiers	→	ID { , ID } ∗
Statements	→	*Statement* ∗
Statement	→	; | *Block* | *Assignment* | *IfStatement*
		| *WhileStatement* | *PrintStatement*
Block	→	'{' *Statements* '}'
Assignment	→	ID = *Expression* ;
IfStatement	→	`if` (*Expression*) *Statement*
		{ `else` *Statement* }?
WhileStatement	→	`while` (*Expression*) *Statement*
PrintStatement	→	`System.out.println` (*Expression*) ;

Jay Concrete Syntax, continued

$$
\begin{array}{rcl}
\mathit{Expression} & \rightarrow & \mathit{Conjunction}\ \{\ \texttt{||}\ \mathit{Conjunction}\ \}* \\
\mathit{Conjunction} & \rightarrow & \mathit{Relation}\ \{\ \texttt{\&\&}\ \mathit{Relation}\ \}* \\
\mathit{Relation} & \rightarrow & \mathit{Addition}\ \{\ \mathit{RelOp}\ \mathit{Addition}\ \}? \\
\mathit{RelOp} & \rightarrow & \texttt{<}\ \mid \texttt{<=}\ \mid \texttt{>}\ \mid \texttt{>=}\ \mid \texttt{==}\ \mid \texttt{!=} \\
\mathit{Addition} & \rightarrow & \mathit{Term}\ \{\ \mathit{AddOp}\ \mathit{Term}\ \}\ * \\
\mathrm{AddOp} & \rightarrow & \texttt{+}\ \mid \texttt{-} \\
\mathit{Term} & \rightarrow & \mathit{Negation}\ \{\ \mathit{MulOp}\ \mathit{Negation}\ \}\ * \\
\mathrm{MulOp} & \rightarrow & `*'\ \mid \texttt{/} \\
\mathit{Negation} & \rightarrow & \mathit{NegOp}?\ \mathit{Factor} \\
\mathit{NegOp} & \rightarrow & \texttt{!}\ \mid \texttt{-} \\
\mathit{Factor} & \rightarrow & \mathrm{ID}\ \mid \mathrm{LITERAL}\ \mid (\ \mathit{Expression}\ )
\end{array}
$$

Jay Abstract Syntax

Program	→	*Declaration*∗ *Statement*
Declaration	→	*Type* ID∗
Statement	→	*Skip* | *Block* | *Assignment* | *Conditional*
		| *Loop* | *Print*
Block	→	*Statement*∗
Assignment	→	ID *Expression*
Conditional	→	*Expression Statement Statement*
Loop	→	*Expression Statement*
Print	→	*Expression*
Expression	→	*Variable* | *IntLitExpr* | *BoolLitExpr*
		| *BinaryExpr* | *UnaryExpr*
Variable	→	ID
IntLitExpr	→	INT_LIT
BoolLitExpr	→	BOOL_LIT
BinaryExpr	→	*Expression* OPERATOR *Expression*
UnaryExpr	→	OPERATOR *Expression*