Context-free grammar for Minijava variant (version komp12.2)

Reserved words are **bold face**. Terminal and non-terminal symbols are *italics*. Literal strings which are not reserved words are in `typewriter face`.

Please see the course project pages for the latest version of this grammar.

Program	$\rightarrow$	*MainClass ClassDecl**
MainClass	$\rightarrow$	**class** *id* { **public static void** `main` (**String** [ ] *id*) { *VarDecl* Stmt** } }
ClassDecl	$\rightarrow$	**class** *id* { *VarDecl* MethodDecl** }
VarDecl	$\rightarrow$	*Type id* ;
MethodDecl	$\rightarrow$	**public** *Type id* (*FormalList*) { *VarDecl* Stmt** **return** *Exp* ; }
FormalList	$\rightarrow$	*Type id FormalRest**
	$\rightarrow$	
FormalRest	$\rightarrow$	, *Type id*
Type	$\rightarrow$	**int** [ ]
	$\rightarrow$	**boolean**
	$\rightarrow$	**int**
	$\rightarrow$	*id*
Stmt	$\rightarrow$	{ *Stmt** }
	$\rightarrow$	**if** (*Exp*) *Stmt* **else** *Stmt*
	$\rightarrow$	**while** (*Exp*) *Stmt*
	$\rightarrow$	**System.out.println** (*Exp*) ;
	$\rightarrow$	*id* = *Exp* ;
	$\rightarrow$	*id* [*Exp*] = *Exp* ;
Exp	$\rightarrow$	*Exp Op Exp*
	$\rightarrow$	*Exp* [*Exp*]
	$\rightarrow$	*Exp* . **length**
	$\rightarrow$	*Exp* . *id* (*ExpList*)
	$\rightarrow$	*int_lit*
	$\rightarrow$	**true**
	$\rightarrow$	**false**
	$\rightarrow$	*id*
	$\rightarrow$	**this**
	$\rightarrow$	**new int** [*Exp*]
	$\rightarrow$	**new** *id* ( )
	$\rightarrow$	! *Exp*
	$\rightarrow$	(*Exp*)
Op	$\rightarrow$	&&
	$\rightarrow$	<
	$\rightarrow$	+
	$\rightarrow$	−
	$\rightarrow$	*
ExpList	$\rightarrow$	*Exp ExpRest**
	$\rightarrow$	
ExpRest	$\rightarrow$	, *Exp*

Grammar extensions

These are *grammar* extensions. For a list of all types of extensions, please see project web pages.

Extension 15p:
$$Stmt \quad \rightarrow \quad \textbf{if} \ (\ Exp \ ) \ Stmt$$

Extension 15p/5p (5p if combined with X86_64 and INT32, else 15p):
$$
\begin{aligned}
Type \quad &\rightarrow \quad \textbf{long [ ]} \\
&\rightarrow \quad \textbf{long} \\
Exp \quad &\rightarrow \quad long\_lit
\end{aligned}
$$

Extension 20p (syntax checks) + 10p/30p (see course project web pages for point rules):
$$ClassDecl \quad \rightarrow \quad \textbf{class} \ id \ \textbf{extends} \ id \ \{ \ VarDecl^* \ MethodDecl^* \ \}$$

Extension 20p. Replace first *Stmt* production:
$$Stmt \quad \rightarrow \quad \{ \ VarDecl^* \ Stmt^* \ \}$$
(Please note that Java does not permit reuse of an identifier in a nested block; we should keep to that restriction for Minijava.)

Extension 1p per operator:
$$
\begin{aligned}
Op \quad &\rightarrow \quad <= \\
&\rightarrow \quad > \\
&\rightarrow \quad >= \\
&\rightarrow \quad == \\
&\rightarrow \quad !=
\end{aligned}
$$

Extension 2p:
$$Op \quad \rightarrow \quad ||$$

Extension Xp (suggest your own extension!)

Lexicals

$$
\begin{aligned}
id \quad &:= \quad \text{[a-zA-Z][a-zA-Z0-9]*} \\
int\_lit \quad &:= \quad 0 \mid \text{[1-9][0-9]*} \\
long\_lit \quad &:= \quad 0 \mid \text{[1-9][0-9]*[lL]}
\end{aligned}
$$

Context rules and Semantics

Minijava does not have method overloading.

The semantics of a Minijava program is defined by Java's semantics.

A program that is invalid Java is also invalid Minijava. Student Minijava compilers do not need to reject Minijava programs with potential variable reads prior to their their first initialisation.