Proyecto

John Doe

June 5, 2025

Contents

Program ::= VarDecList ClassDecList FunDecList
VarDecList ::= (VarDec)*
ClassDecList ::= (ClassDec)*
FunDecList ::= (FunDec)+
FunDec ::= fun id([ParamDecList]) **:** Type { Body }
Body ::= VarDecList StmtList
ParamDecList ::= id : Type (, id: Type)*
VarDec ::= [val | var] id : Type [= CExp];
ListDec ::= [val | var] id : [MutableList | List]<Type> [= ListExp];
ListExp ::= [listOf | mutableListOf] **(** (CExp)* **)**
ClassDec ::= **class** id(Arguments) **{** VarDecList FunDecList **}**
Arguments ::= val id : Type (, val id: Type)*
ListDec ::= [val | var] : Type id [= CExp];
Type ::= id
VarList :: id (,id)*
StmtList ::= Stmt (; Stmt)*
Stmt ::= id = CExp |
[println | print] (CExp) |
if CExp **{** Body **}** [else **{** Body *}*] |
while **(** CExp **)** **{** Body **}** |
do **{** Body **}** while **(** CExp **)** |
for (id in [[CExp [..[<] | downTo] CExp] | ListExp]) **{** Body **}** |
return ([CExp]) |
when [*(* CExp **)*] *{** (CExp -> Stmt)* **else** -> Stmt **}** |
CExp ::= Exp [(<|<=|==) Exp]
Exp ::= Term ((+ | -) Term)*
Term ::= Factor ((|/) **Factor)**

Factor ::= id | Num | Bool | (Exp) |
if **(** CExp **)** CExp **else** CExp) | id ([ArgList]) |
when [*(* CExp **)*] *{** (CExp -> CExp)* **else** -> CExp **}** |
ListExp[CExp]
ArgList ::= CExp (, CExp)*
Bool ::= true | false