**First:**

* -First (Program)= {@,^}
* -first (Start-Symbols)={@,^}
* -first(End-Symbols)={$,#}
* -f(ClassDeclaration)={ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational }
* -f(ClassDeclaration’)={{,Infer}
* -f(Type)={ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational}
* -f(Class\_Implementation)={ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational ,em,<,Require,ID,\*}
* -f(Temp)={ ( , , , em }
* -f(Variable\_decl)={ , ,em}
* -f(Variable\_decl’)={;,[}
* -f(Method\_decl)={( }
* -f(Method\_decl’)={;,{}
* -f(Comment)={<,\*}
* -f(require\_command)={ Require }
* -f(Func\_Call)={ ID}
* -f(Func Decl)={ ( }
* -f(ParameterList)={ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational ,em,None}
* -f(Non-Empty List)={ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational }
* -f(Non-Empty List-)={ ,,em }
* -f(ID\_List)={,}
* -f(ID\_List-)={,,em}
* -F(Statements)={ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational ,em, if, However, when, Respondwith, Endthis , return }
* -f(Statement)={ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis, return }
* -f(Assignment)={ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational }
* -f(If \_Statement)={ if}
* -f(If \_Statement’)={else,em}
* -f(However \_Statement)={ However}
* -f(when\_Statement)={ when}
* -f(Respondwith \_ Statement)={ Respondwith,return }
* -f( Endthis \_Statement)={ Endthis }
* -f(Argument\_List)={ ID,Number ,em}
* -f(NonEmpty\_Argument\_List)={ ID,Number }
* -f(NonEmpty\_Argument\_List-)={ ,,em }
* -f(Expression)={ ID,Number}
* -f(Expression-)={+,-,em}
* -f(Term)={ ID,Number }
* -f(Term-)={\*,/,em}
* -f(Add\_Op)={+,-}
* -f(factor)={ID,Number}
* -f(Multi\_Op)={\*,/}
* -f(Block Statements)={{}
* -f(Condition \_Expression)={ ID,Number }
* -f(Condition \_Expression’)={&,|,em}
* -f(Condition)={ ID,Number }
* -f(Condition \_Op)={&,|}
* -f(Comparison \_Op)= {=,!,<,>}
* -f(Comparison \_Op’)={>,<}
* -f(Comparison \_Op’’)={=,em}
* -f(F\_name)={ID}

**Follow:**

* -Follow(Program)={$}
* =Follow(Start-Symbols)={ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational }
* -Follow(End-Symbols)={$}
* -f(ClassDeclaration)={ $ , #}
* -f(ClassDeclaration’)={$ , #}
* -f(Type)={ ID}
* -f(Class\_Implementation)={ }}
* -f(Temp)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational ,<,Require,ID,\* , } }
* -f(Variable\_decl)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis , return , <,Require,ID,\*, = , } }
* -f(Variable\_decl’)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational ,<,Require,ID,\*, if, However, when, Respondwith, Endthis , return , = , } }
* -f(Method\_decl)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational ,<,Require,ID,\* , } }
* -f(Method\_decl’)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational ,<,Require,ID,\* , } }
* -f(Comment)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational ,<,Require,ID,\*, } }
* -f(Require\_command)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational ,<,Require,ID,\* , } }
* -f(Func\_Call)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational ,<,Require,ID,\* , } }
* -f(Func Decl)={ ;,{}
* -f(ParameterList)={ )}
* -f(Non-Empty List)={ ) }
* -f(Non-Empty List-)={ )}
* -f(ID\_List)= { ;,[ }
* -f(ID\_List-)= { ;,[ }
* -F(Statements)={ } }
* -f(Statement)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis , return , } }
* -f(Assignment)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis , return , } }
* -f(If \_Statement)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis , return , } }
* -f(If \_Statement’)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis , return , } }
* -f(However \_Statement)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis , return , } }
* -f(when\_Statement)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis , return , } }
* -f(Respondwith \_ Statement)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis , return , } }
* -f( Endthis \_Statement)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis , return , } }
* -f(Argument\_List)={ )}
* -f(NonEmpty\_Argument\_List)={ ) }
* -f(NonEmpty\_Argument\_List-)={ ) }
* -f(Expression)={ ; , , , =,!,<,> , & , | , ) }
* -f(Expression-)= { ; , , , =,!,<,> , & , | , ) }
* -f(term)= {+,-, ; , , , =,!,<,> , & , | , )}
* -f(term-)={+,-, ; , , , =,!,<,> , & , | , ) }
* -f(Add\_Op)={ ID,Number }
* -f(factor)={ \*,/,+,-,),;,=,!,<,>,&,| }
* -f(Multi\_Op)={ ID,Number }
* -f(Block Statements)= { Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless,Rational , if, However, when, Respondwith, Endthis , return , else, } }
* -f(Condition \_Expression)={ )}
* -f(Condition \_Expression’)={ )}
* -f(Condition)= {&,| , )}
* -f(Condition \_Op)={ ID,Number }
* -f(Comparison \_Op)={ ID,Number }
* -f(Comparison \_Op’)={ ID,Number }
* -f(Comparison \_Op’’)={ ID,Number }
* -f(F\_name)={.}

New Grammar:

- Non-Empty List →Type ID Non-Empty List’

Non-Empty List’ → ,Type ID Non-Empty List’|em

-ID\_List →ID ID\_List’

ID\_List‘→ , ID ID\_List ‘|em

- NonEmpty\_Argument\_List →Expression NonEmpty\_Argument\_List’

NonEmpty\_Argument\_List’→ ,Expression NonEmpty\_Argument\_List’|em

-Expression→Term Expression’

Expression’→Add\_Op Term Expression’|em

- Term →Factor Term’

Term’→Multi\_Op Factor Term’|em

-ClassDeclaration→Type ID ClassDeclaration’

ClassDeclaration’→{ Class\_Implementation}|Infer { Class\_Implementation}

- Class\_ImplementationàComment Class\_Implementation|require\_command Class\_Implementation|Func\_Call Class\_Implementation|em| Class\_Implementation’

Class\_Implementation’à

-Variable\_Decl→Type ID\_List Variable\_Decl’|em

Variable\_Decl’ à ; Variable\_Decl | [ID] ; Variable\_Decl

- Method\_Declà Func Decl Method\_Decl’

Method\_Decl’à;| { Variable\_Decl Statements }

- If \_Statementà if (Condition \_Expression) Block Statements If \_Statement’

If \_Statement’à else Block Statements|em

- Condition \_Expressionà Condition Condition \_Expression’

Condition \_Expression’à Condition \_Op Condition|em

- Comparison \_Opà==|!=| Comparison \_Op’

Comparison \_Op’à> Comparison \_Op ‘’|< Comparison \_Op’’

Comparison \_Op’’à=|em