Original Grammar

Program ::= *Decl*$^+$

Decl ::= *VariableDecl* | *FunctionDecl* | *ConstDecl* | *ClassDecl* | *IntefaceDecl*

VariableDecl ::= *Variable* **;**

Variable ::= *Type* **ident**

ConstDecl ::= **static** *ConstType* **ident** *;*

ConstType ::= **int** | **double** | **boolean** | **string**

Type ::= **int** | **double** | **boolean** | **string** | **ident** | *Type***[ ]**

FunctionDecl ::= *Type* **ident (** *Formals* **)** *StmtBlock* | **void ident (** *Formals* **)** *StmtBlock*

Formals ::= *Variable* **,** *Formals* | *Variable*

ClassDecl ::= **class ident** < **extends ident**> < **implements ident**$^+$ **,** > **{** *Field** **}**

Field ::= *VariableDecl* | *FunctionDecl* | *ConstDecl*

InterfaceDecl ::= **interface ident {** *Prototype** **}**

Prototype ::= *Type* **ident (** *Formals* **) ;** | **void ident (** *Formals* **) ;**

StmtBlock ::= **{** *VariableDecl** *ConstDecl** *Stmt** **}**

Stmt ::= < *Expr* > **;** | *IfStmt* | *WhileStmt* | *ForStmt* | *BreakStmt* | *ReturnStmt* | *PrintStmt*
| *StmtBlock*

IfStmt ::= **if (** *Expr* **)** *Stmt* < **else** *Stmt* >

WhileStmt ::= **while (** *Expr* **)** *Stmt*

ForStmt ::= **for (** *Expr* **;** *Expr* **;** *Expr* **)** *Stmt*

ReturnStmt ::= **return** *Expr* **;**

BreakStmt ::= **break ;**

PrintStmt ::= **System.out.println (** *Expr*$^+$ **,) ;**

Expr ::= *LValue* **=** *Expr* | *Constant* | *LValue* | **this** | **(** *Expr* **)** | *Expr* **-** *Expr* | *Expr* **/** *Expr*
| *Expr* **%** *Expr* | **-** *Expr* | *Expr* **>** *Expr* | *Expr* **>=** *Expr* | *Expr* **!=** *Expr* | *Expr* **||** *Expr*
| **!** *Expr* | **New (ident)**

LValue ::= **ident** | *Expr* **. ident**

Constant ::= **intConstant** | **doubleConstant** | **booleanConstant** | **stringConstant** | **null**

Expanded Grammar

Init → *Program*

1. *Program* → *Decl Program*
2. *Program* → *Decl*
3. *Decl* → *VariableDecl*
4. *Decl* → *FunctionDecl*
5. *Decl* → *ConstDecl*
6. *Decl* → *ClassDecl*
7. *Decl* → *InterfaceDecl*
8. *VariableDecl* → *Variable* **;**
9. *Variable* → *Type* **ident**
10. *ConstDecl* → **static** *ConstType* **ident ;**
11. *ConstType* → **int**
12. *ConstType* → **double**
13. *ConstType* → **boolean**
14. *ConstType* → **string**
15. *Type* → **int** *TypeArray*
16. *Type* → ***double*** *TypeArray*
17. *Type* → **boolean** *TypeArray*
18. *Type* → **string** *TypeArray*
19. *Type* → **ident** *TypeArray*
20. *TypeArray* → **[]** *TypeArray*
21. *TypeArray* → *ε*
22. *FunctionDecl* → *Type* **ident (** *Formals* **)** *StmtBlock*
23. *FunctionDecl* → **void ident (** *Formals* **)** *StmtBlock*
24. *Formals* → *Variable* **,** *Formals*
25. *Formals* → *Variable*
26. *ClassDecl* → **class ident** *Extends Implements* **{** *FieldStar* **}**
27. *Extends* → **extends ident**
28. *Extends* → *ε*
29. *Implements* → **implements ident** *ImplementsIdentPlus*
30. *Implements* → *ε*
31. *ImplementsIdentPlus* → **, ident** *ImplementsIdentPlus*
32. *ImplementsIdentPlus* → *ε*
33. *FieldStar* → *Field FieldStar*
34. *FieldStar* → *ε*
35. *Field* → *VariableDecl*
36. *Field* → *FunctionDecl*
37. *Field* → *ConstDecl*
38. *InterfaceDecl* → **interface ident {** *PrototypeStar* **}**
39. *PrototypeStar* → *Prototype PrototypeStar*
40. *PrototypeStar* → *ε*
41. *Prototype* → *Type* **ident (** *Formals* **) ;**
42. *Prototype* → **void ident (** *Formals* **) ;**
43. *StmtBlock* → **{** *StmtBlockDeclStar* **}**
44. *StmtBlockDeclStar* → *StmtBlockDecl StmtBlockDeclStar*
45. *StmtBlockDeclStar* → *ε*
46. *StmtBlockDecl* → *VariableDecl*
47. *StmtBlockDecl* → *ConstDecl*
48. *StmtBlockDecl* → *Stmt*

49. *Stmt* → *OpenStmt*
50. *Stmt* → *ClosedStmt*
51. *OpenStmt* → **if (** *Expr* **)** *Stmt*
52. *OpenStmt* → **if (** *Expr* **)** *ClosedStmt* **else** *OpenStmt*
53. *OpenStmt* → **for (** *Expr* **;** *Expr* **;** *Expr* **)** *OpenStmt*
54. *OpenStmt* → **while (** *Expr* **)** *OpenStmt*
55. *ClosedStmt* → *SimpleStatemet*
56. *ClosedStmt* → **if (** *Expr* **)** *ClosedStmt* **else** *ClosedStmt*
57. *ClosedStmt* → **for (** *Expr* **;** *Expr* **;** *Expr* **)** *ClosedStmt*
58. *ClosedStmt* → **while (** *Expr* **)** *ClosedStmt*
59. *SimpleStatemet* → *Expr* **;**
60. *SimpleStatemet* → **;**
61. *SimpleStatemet* → *BreakStmt*
62. *SimpleStatemet* → *ReturnStmt*
63. *SimpleStatemet* → *PrintStmt*
64. *SimpleStatemet* → *StmtBlock*
65. *SimpleStatemet* → *CallStmt*
66. *ReturnStmt* → **return** *Expr* **;**
67. *BreakStmt* → **break ;**
68. *PrintStmt* → **System . out . println (** *ExprPlus* **)**
69. *ExprPlus* → *Expr* **,** *ExprPlus*
70. *ExprPlus* → *Expr*
71. *CallStmt* → **ident (** *Actuals* **)**
72. *CallStmt* → **ident . ident (** *Actuals* **)**
73. *Actuals* → *Expr* **,** *Actuals*
74. *Actuals* → *Expr*
75. *Expr* → **ident** *Access* **=** *ExprSubLevel1*
76. *Expr* → *ExprSubLevel1*
77. *ExprSubLevel1* → *ExprSubLevel1* **||** *ExprSubLevel2*
78. *ExprSubLevel1* → *ExprSubLevel2*
79. *ExprSubLevel2* → *ExprSubLevel2* **!=** *ExprSubLevel3*
80. *ExprSubLevel2* → *ExprSubLevel3*
81. *ExprSubLevel3* → *ExprSubLevel3* **>** *ExprSubLevel4*
82. *ExprSubLevel3* → *ExprSubLevel3* **>=** *ExprSubLevel4*
83. *ExprSubLevel3* → *ExprSubLevel4*
84. *ExprSubLevel4* → *ExprSubLevel4* **-** *ExprSubLevel5*
85. *ExprSubLevel4* → *ExprSubLevel5*
86. *ExprSubLevel5* → *ExprSubLevel5* **/** *ExprSubLevel6*
87. *ExprSubLevel5* → *ExprSubLevel5* **%** *ExprSubLevel6*
88. *ExprSubLevel5* → *ExprSubLevel6*
89. *ExprSubLevel6* → **New (ident)**
90. *ExprSubLevel6* → *ExprSubLevel7*
91. *ExprSubLevel7* → **-** *ExprSubLevel8*
92. *ExprSubLevel7* → **!** *ExprSubLevel8*
93. *ExprSubLevel7* → *ExprSubLevel8*
94. *ExprSubLevel8* → **(** *Expr* **)**
95. *ExprSubLevel8* → **this**
96. *ExprSubLevel8* → **intConstant**
97. *ExprSubLevel8* → **doubleConstant**
98. *ExprSubLevel8* → **booleanConstant**
99. *ExprSubLevel8* → **stringConstant**

100. *ExprSubLevel8* → **null**
101. *ExprSubLevel8* → **ident** *Access*
102. *Access* → **. ident** *Access*
103. *Access* → ε

NOTE: Bold text are the terminals of the grammar.

First and Follow

Nonterminal	FIRST	FOLLOW
Init	{static,class,interface,int,double,boolean ,string,ident,void}	$
Program	{static,class,interface,int,double,boolean ,string,ident,void}	$
DeclAdditional	{static,int,double,boolean,string,ident,vo id}	static,class,interface,int,double,boolean,string ,ident,void,$,}
Decl	{static,class,interface,int,double,boolean ,string,ident,void}	static,class,interface,int,double,boolean,string ,ident,void,$
ConstType	{int,double,boolean,string}	ident,[]
Type	{int,double,boolean,string,ident}	ident,[]
FuncProtoInit	{int,double,boolean,string,ident,void}	ident
Formals	{int,double,boolean,string,ident}	)
Extends	{extends,"}	static,class,interface,int,double,boolean,string ,ident,void,$,{
Implements	{",ident}	{,ident
ImplementsIdentPlus	{,,"}	{,ident
Field	{static,",int,double,boolean,string,ident,v oid}	}
Prototype	{int,double,boolean,string,ident,void,"}	}
StmtBlock	{{}	},;,if,while,for,break,return,System,{,ident,Ne w,-,!,(,this,intConstant,doubleConstant,boole anConstant,stringConstant,null,else,static,cla ss,interface,int,double,boolean,string,void,$
VariableDeclStar	{int,double,boolean,string,ident,"}	static,;,if,while,for,break,return,System,{,ident ,New,-,!,(,this,intConstant,doubleConstant,bo oleanConstant,stringConstant,null,},else,clas s,interface,int,double,boolean,string,void,$
ConstDeclStar	{static,"}	;,if,while,for,break,return,System,{,ident,New, -,!,(,this,intConstant,doubleConstant,boolean Constant,stringConstant,null,},else,static,clas s,interface,int,double,boolean,string,void,$
StmtStar	{",;,if,while,for,break,return,System,{,ide	}

	nt,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null}	
Stmt	{;,if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null}	},;,if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else
ElseStmt	{else,''}	},;,if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else
PrintStmtExpr	{,,''}	)
Expr	{ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null}	;,),,,},if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else
ExprSubLevel1	{New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,ident}	;,),,,},if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else,||
ExprSubLevel2	{New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,ident}	;,),,,},if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else,||,!=
ExprSubLevel3	{New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,ident}	;,),,,},if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else,||,!=,>,>=
ExprSubLevel4	{New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,ident}	;,),,,},if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else,||,!=,>,>=
ExprSubLevel5	{New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,ident}	;,),,,},if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else,||,!=,>,>=,/,%
ExprSubLevel6	{New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,ident}	;,),,,},if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else,||,!=,>,>=,/,%
ExprSubLevel7	{-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,ident}	;,),,,},if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else,||,!=,>,>=,/,%
ExprSubLevel8	{(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,ident}	;,),,,},if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,bool

		eanConstant,stringConstant,null,else,||,!=,>,>=,/,%
Access	{.,''}	=,;,),,,},if,while,for,break,return,System,{,ident,New,-,!,(,this,intConstant,doubleConstant,booleanConstant,stringConstant,null,else,||,!=,>,>=,/,%

Parsing Table

In the parsing table file.