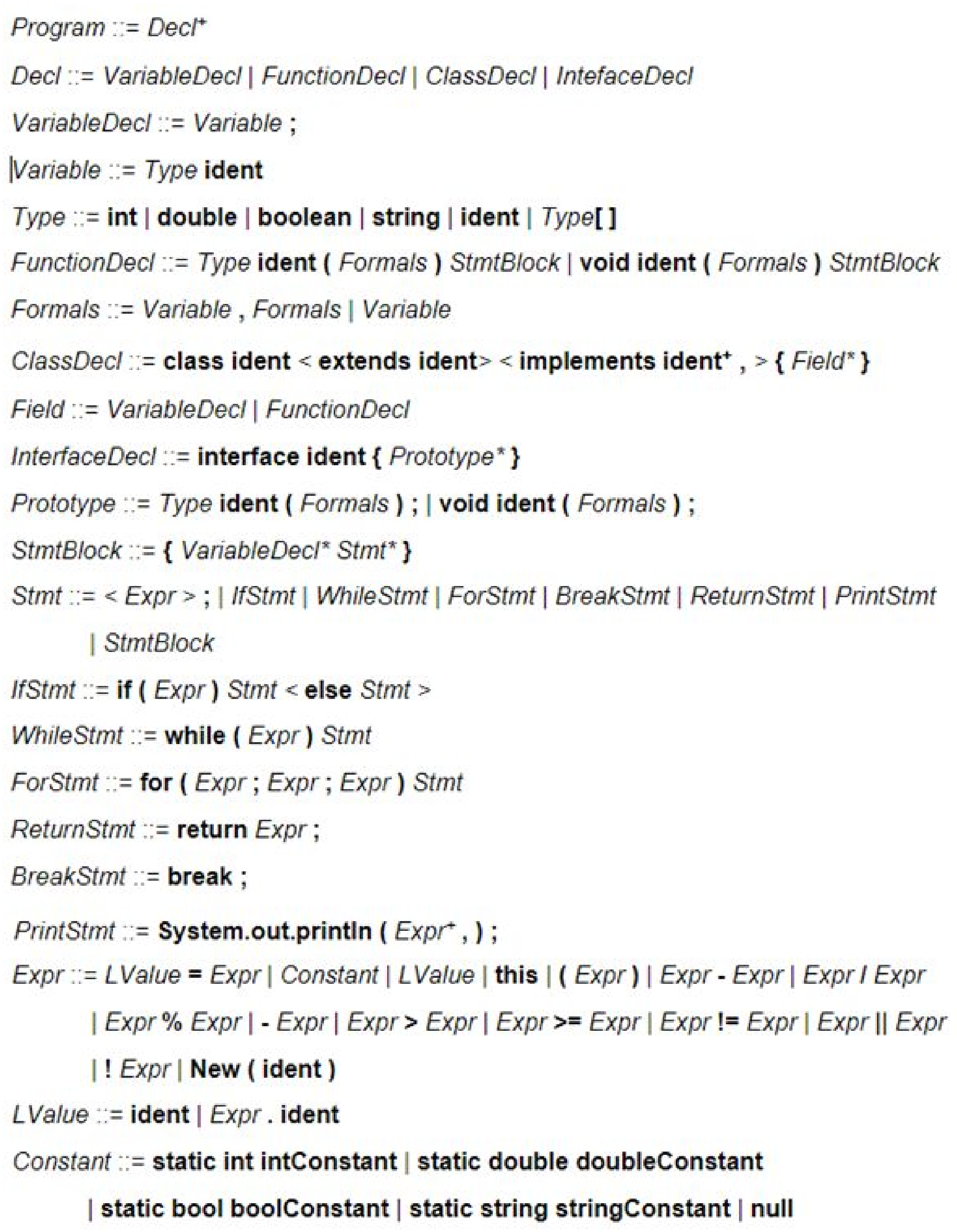

Program ::= *Decl*$^{+}$

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

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

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

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*

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

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

StmtBlock ::= **{** *VariableDecl** *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 ::= **static int intConstant** | **static double doubleConstant**
| **static bool boolConstant** | **static string stringConstant** | **null**

Generalidades de la gramática:

- **x** significa que x es un terminal, un token devuelto por el analizador léxico. Los terminales están en minúscula y negrita excepto aquellos que son palabras reservadas
- *x* en itálico es un no terminal. Todos los no terminales tienen la primera letra en mayúscula
- <x> significa cero o una ocurrencia de x, es decir, x es opcional
- x* significa cero o más ocurrencias de x
- x^{+} significa una o más ocurrencias de x (Listadas, seguidas por coma excepto el último elemento)
- | significa las alternativas de las producciones

[Segunda versión]

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**

[Gramática Original]

Program ::= Decl+
Decl ::= VariableDecl I FunctionDecl I ClassDecl I IntefaceDecl
VariableDecl ::= Variable **;**
Variable ::= Type **ident**
Type ::= **int** I **double** I **boolean** I **string** I **ident** I Type **[]**
FunctionDecl ::= Type **ident (** Formals **)** StmtBlock I **void ident (** Formals **)** StmtBlock
Formals ::= Variable **,** Formals I Variable
ClassDecl ::= **class ident** < **extends ident** > < **implements ident+ ,** > **{** Field* **}**
Field ::= VariableDecl I FunctionDecl
InterfaceDecl ::= **interface ident {** Prototype* **}**
Prototype ::= Type **ident (** Formals **) ;** I **void ident (** Formals **) ;**
StmtBlock ::= **{** VariableDecl* Stint* **}**
Stmt ::= < Expr > **;** I IfStmt I WhileStmt I ForStmt I BreakStmt I ReturnStmt I 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 I *Constant* I LValue I **this** I **(** Expr**)** I Expr **-** Expr I *Expr **I** Expr* | Expr **%** Expr I **-** Expr I Expr **>** Expr I Expr **>=** Expr I Expr **I=** Expr I Expr **II** Expr | **!** Expr I **New (ident)**
LValue ::= **ident** I Expr. **ident**
Constant ::= **static int intConstant** I **static double doubleConstant** | **static bool boolConstant** I **static string stringConstant** I **null**

[Gramática Modificada]

Program’ ::=	Program
Program ::=	Decl
Decl ::=	FunctionDecl Decl’
Decl ::=	ClassDecl Decl’
Decl ::=	InterfaceDecl Decl’
Decl ::=	VariableDecl Decl’
Decl’ ::=	Decl
Decl’ ::=	ε
VariableDecl ::=	Variable **;** EOF
Variable ::=	Type **ident**
Type ::=	**int**
Type ::=	**double**
Type ::=	**boolean**
Type ::=	**string**
Type ::=	**ident**
Type ::=	Type **[ ]**
FunctionDecl ::=	Type **ident (** Formals **)** StmtBlock
FunctionDecl ::=	**void ident (** Formals **)** StmtBlock
Formals ::=	Variable **,** Formals
Formals ::=	Variable
ClassDecl ::=	**class ident** ClassDecl1 classDecl2 **{** Field **}**
ClassDecl1 ::=	**extends ident**
ClassDecl1 ::=	ε
ClassDecl2 ::=	**implements ident** ClassDecl3
ClassDecl2 ::=	ε
ClassDecl3 ::=	, **ident** ClassDecl3
ClassDecl3::=	ε
Field ::=	VariableDecl Field
Field ::=	InterfaceDecl Field
Field ::=	ε
InterfaceDecl ::=	**interface ident {** Prototype **}**
Prototype ::=	Type **ident (** Formals **) ;** Prototype
Prototype ::=	**void ident (** Formals **) ;** Prototype
Prototype ::=	ε
StmtBlock ::=	**{** StmtBlock1 StmtBlock2 **}**
StmtBlock1 ::=	VariableDecl StmtBlock1
StmtBlock1 ::=	ε
StmtBlock2 ::=	Stmt StmtBlock2
StmtBlock2 ::=	ε
Stmt ::=	**;**
Stmt ::=	Expr **;**
Stmt ::=	IfStmt
Stmt ::=	WhileStmt
Stmt ::=	ForStmt
Stmt ::=	BreakStmt

Stmt ::=	ReturnStmt
Stmt ::=	PrintStmt
Stmt ::=	StmtBlock
IfStmt ::=	**if (** Expr **)** Stmt ElseStmt
ElseStmt ::=	**else** Stmt
ElseStmt ::=	ε
WhileStmt ::=	**while (** Expr **)** Stmt
ForStmt ::=	**for (** Expr **;** Expr **;** Expr **)** Stmt
ReturnStmt ::=	**return** Expr **;**
BreakStmt ::=	**break ;**
PrintStmt ::=	**System.out.println (** PrintStmt2 **) ;**
PrintStmt2::=	Expr PrintStmt3
PrintStmt3::=	**,** Expr PrintStmt3
PrintStmt3::=	ε
Expr ::=	LValue **=** RValue
RValue ::=	**New (ident)**
RValue ::=	Expr
Expr ::=	**this**
Expr ::=	ExprLogi
Expr ::=	**-** Expr
Expr ::=	**!** Expr
ExprLogi ::=	ExprDiv
ExprLogi ::=	Expr **>** ExprLogi
ExprLogi ::=	Expr **>=** ExprLogi
ExprLogi ::=	Expr **!=** ExprLogi
ExprLogi ::=	Expr **||** ExprLogi
ExprDiv ::=	ExprMin
ExprDiv ::=	Expr **%** ExprMin
ExprDiv ::=	Expr **/** ExprMin
ExprMin ::=	Factor **-** ExprMin
ExprMin ::=	Factor
Factor ::=	Constant
Factor ::=	LValue
Factor ::=	**(** Expr **)**
LValue ::=	**ident**
LValue ::=	Expr **. ident**
Constant ::=	**static int intConstant**
Constant ::=	**static double doubleConstant**
Constant ::=	**static bool boolConstant**
Constant ::=	**static string stringConstant**
Constant ::=	**null**

[Gramática Modificada V2]

*Inicio' ::=	Program
*Program ::=	Decl
*Decl ::=	Variable DECLARACION Decl1
DECLARACION ::=	;
DECLARACION ::=	FunctionDecl

*Decl ::=	ClassDecl Decl1
*Decl ::=	InterfaceDecl Decl1
*Decl ::=	ConstDecl Decl1
*Decl ::=	FunctionDecl1 Decl1

*Decl1 ::=	Decl
*Decl1 ::=	ε
*VariableDecl ::=	Variable **;**
*Variable ::=	Type TypeArray **ident**
*ConstDecl ::=	**static** ConstType **ident ;**
*ConstType ::=	**int**
*ConstType ::=	**double**
*ConstType ::=	**boolean**
*ConstType ::=	**string**
*Type ::=	ConstType
*Type ::=	**ident**
***TypeArray**: :=	**[ ] TypeArray**
***TypeArray**: :=	ε

*FunctionDecl ::=	Variable **(** Formals **)** StmtBlock
FunctionDecl1 ::=	**void ident (** Formals **)** StmtBlock
*Formals ::=	Variable **,** Formals
*Formals ::=	Variable
*ClassDecl ::=	**class ident** ClassDecl1 classDecl2 **{** Field **}**
*ClassDecl1 ::=	**extends ident**
*ClassDecl1 ::=	ε
*ClassDecl2 ::=	**implements ident** ClassDecl3
*ClassDecl2 ::=	ε
*ClassDecl3 ::=	, **ident** ClassDecl3
*ClassDecl3::=	ε

*Field ::=	VariableDecl Field
*Field ::=	FunctionDeclField
*Field ::=	ConstDeclField
*Field ::=	ε
*InterfaceDecl ::=	**interface ident {** Prototype **}**
*Prototype ::=	Type TypeArray **ident (** Formals **) ;** Prototype
*Prototype ::=	**void ident (** Formals **) ;** Prototype
*Prototype ::=	ε
*StmtBlock ::=	**{** StmtBlock1 StmtBlock2 **}**

*StmtBlock1 ::=	VariableDecl StmtBlock1
*StmtBlock1 ::=	ε
*StmtBlock2 ::=	Stmt StmtBlock2
*StmtBlock2 ::=	ε
*Stmt ::=	**;**
*Stmt ::=	,Expr **;**
*Stmt ::=	IfStmt
*Stmt ::=	WhileStmt
*Stmt ::=	ForStmt
*Stmt ::=	BreakStmt
*Stmt ::=	ReturnStmt
*Stmt ::=	PrintStmt
*Stmt ::=	StmtBlock
*IfStmt ::=	**if (** Expr **)** Stmt ElseStmt
*ElseStmt ::=	**else** Stmt
*ElseStmt ::=	ε
*WhileStmt ::=	**while (** Expr **)** Stmt
*ForStmt ::=	**for (** Expr **;** Expr **;** Expr **)** Stmt
*ReturnStmt ::=	**return** Expr **;**
*BreakStmt ::=	**break ;**
*PrintStmt ::=	**System.out.println (** PrintStmt2 **) ;**
*PrintStmt2::=	Expr PrintStmt3
*PrintStmt3::=	**,** Expr PrintStmt3
*PrintStmt3::=	ε
*RValue ::=	**New (ident)**
*RValue ::=	Expr
*Expr ::=	A Factor Expr1
Expr1::=	Operacion Expr
Expr1::=	ε
*A :==	!
*A :==	**-**
*A :==	ε
Operacion ::=	=
Operacion ::=	>
Operacion ::=	>=
Operacion ::=	!=
Operacion ::=	||
Operacion ::=	%
Operacion ::=	/
Operacion ::=	**-**
*Factor ::=	Constant
*Factor ::=	LValue
*Factor ::=	**(** Expr **)**
*LValue ::=	**ident** LValue1
*LValue ::=	this **. ident**
*LValue1 ::=	**.ident** LValue1
*LValue1 ::=	ε
*Constant ::=	**intConstant**

*Constant	::=	**doubleConstant**
*Constant	::=	**boolConstant**
*Constant	::=	**stringConstant**
*Constant	::=	**null**