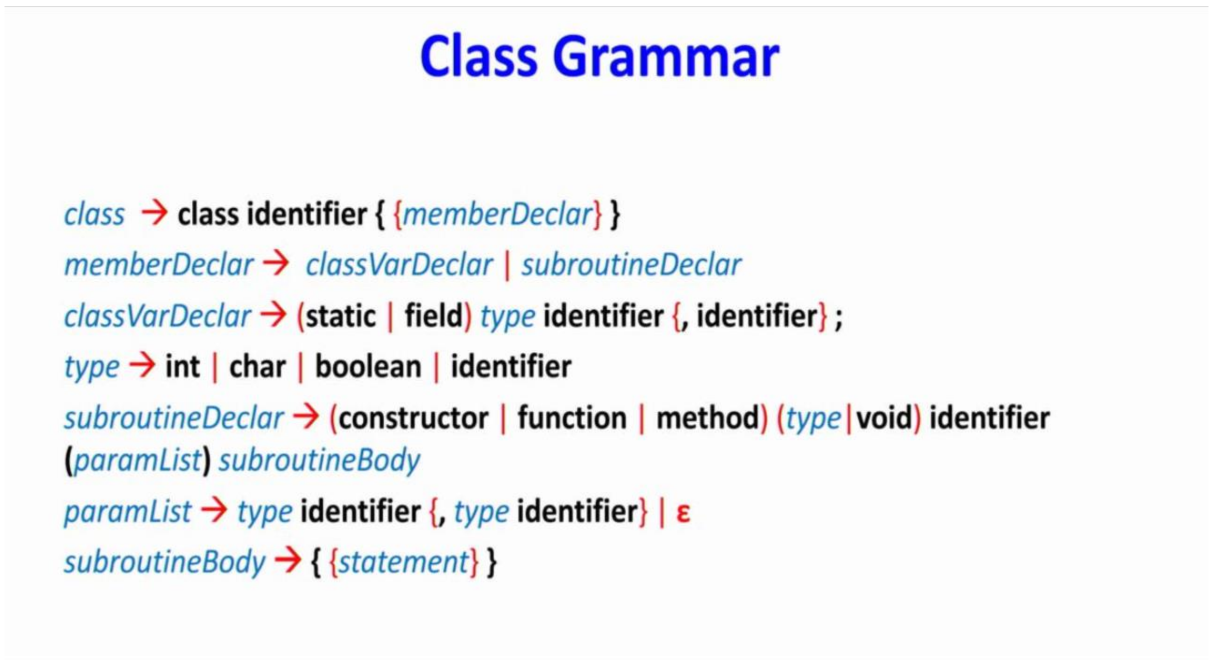


Class Grammar

class → **class identifier {** {*memberDeclar*} **}**

memberDeclar → *classVarDeclar* | *subroutineDeclar*

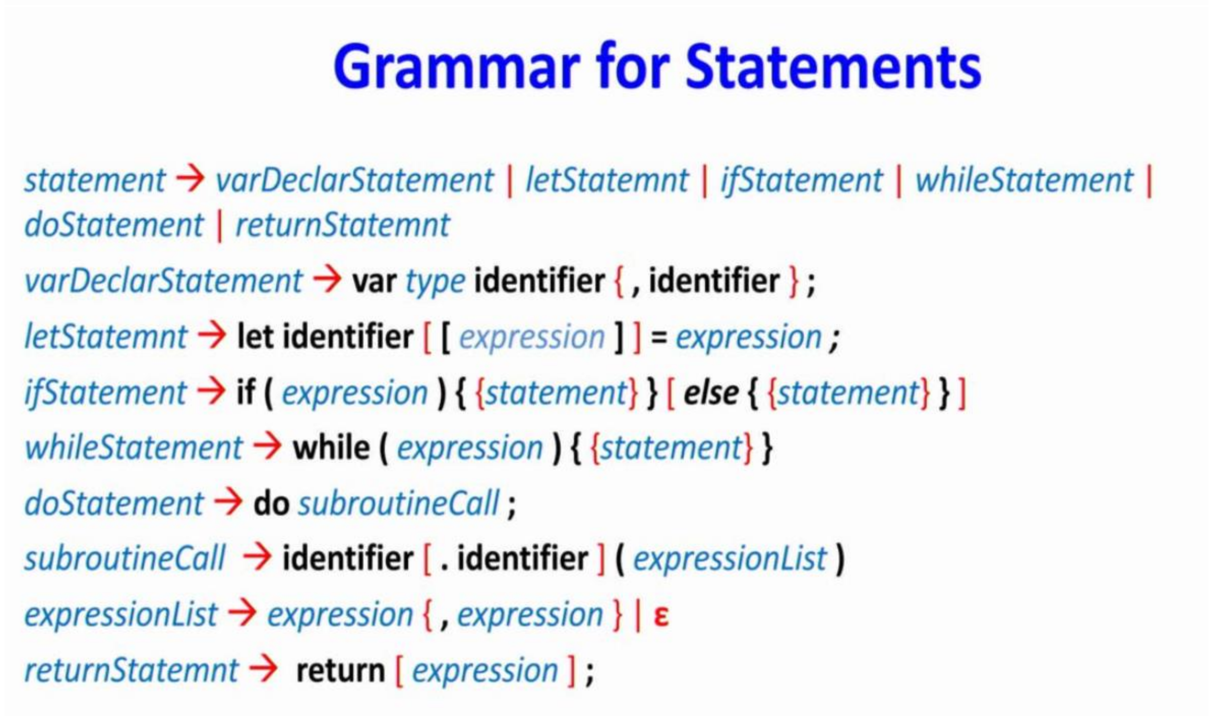
classVarDeclar → (**static** | **field**) *type* **identifier** {**, identifier**} **;**

type → **int** | **char** | **boolean** | **identifier**

subroutineDeclar → (**constructor** | **function** | **method**) (*type*|**void**) **identifier**
(*paramList***)** *subroutineBody*

paramList → *type* **identifier** {**,** *type* **identifier**} | **ε**

subroutineBody → **{** {*statement*} **}**



Grammar for Statements

statement → *varDeclarStatement* | *letStatemnt* | *ifStatement* | *whileStatement* |
doStatement | *returnStatemnt*

varDeclarStatement → **var** *type* **identifier** { **, identifier** } **;**

letStatemnt → **let identifier** [**[** *expression* **]**] **=** *expression* **;**

ifStatement → **if (** *expression* **) {** {*statement*} **}** [***else*** **{** {*statement*} **}**]

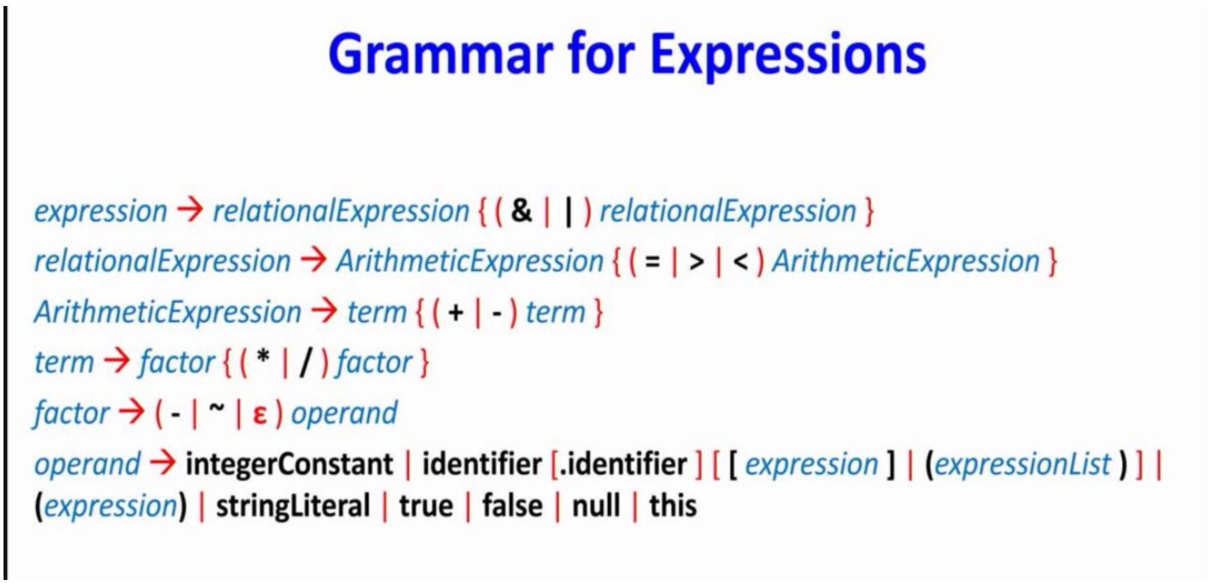
whileStatement → **while (** *expression* **) {** {*statement*} **}**

doStatement → **do** *subroutineCall* **;**

subroutineCall → **identifier** [**. identifier**] **(** *expressionList* **)**

expressionList → *expression* { **,** *expression* } | **ε**

returnStatemnt → **return** [*expression*] **;**



Grammar for Expressions

expression → *relationalExpression* { (**&** | **|**) *relationalExpression* }
relationalExpression → *ArithmeticExpression* { (**=** | **>** | **<**) *ArithmeticExpression* }
ArithmeticExpression → *term* { (**+** | **-**) *term* }
term → *factor* { (***** | **/**) *factor* }
factor → (**-** | **~** | **ε**) *operand*
operand → **integerConstant** | **identifier** [**.identifier**] [**[** *expression* **]** | **(***expressionList* **)**] |
(*expression***)** | **stringLiteral** | **true** | **false** | **null** | **this**

Full Jack Grammar

*classDecla*r → **class identifier {** {*memberDeclar*} **}**
memberDeclar → *classVarDeclar* | *subroutineDeclar*
classVarDeclar → (**static** | **field**) *type* **identifier** {**,** **identifier**} **;**
type → **int** | **char** | **boolean** | **identifier**
subroutineDeclar → (**constructor** | **function** | **method**) (*type* | **void**) **identifier (***paramList***)** *subroutineBody*
paramList → *type* **identifier** {**,** *type* **identifier**} | **ε**
subroutineBody → **{** {*statement*} **}**
statement → *varDeclarStatement* | *letStatemnt* | *ifStatement* | *whileStatement* | *doStatement* | *returnStatemnt*
varDeclarStatement → **var** *type* **identifier** { **, identifier** } **;**
letStatemnt → **let identifier** [**[** *expression* **]**] **=** *expression* **;**
ifStatement → **if (** *expression* **) {** {*statement*} **}** [***else*** **{** {*statement*} **}**]
whileStatement → **while (** *expression* **) {** {*statement*} **}**
doStatement → **do** *subroutineCall* **;**
subroutineCall → **identifier** [**. identifier**] **(** *expressionList* **)**
expressionList → *expression* { **,** *expression* } | **ε**
returnStatemnt → **return** [*expression*] **;**
expression → *relationalExpression* { (**&** | **|**) *relationalExpression* }
relationalExpression → *ArithmeticExpression* { (**=** | **>** | **<**) *ArithmeticExpression* }
ArithmeticExpression → *term* { (**+** | **-**) *term* }
term → *factor* { (***** | **/**) *factor* }
factor → (**-** | **~** | **ε**) *operand*
operand → **integerConstant** | **identifier** [**.identifier**] [**[** *expression* **]** | **(***expressionList* **)**] | **(***expression***)** | **stringLiteral** | **true** | **false** | **null** | **this**