目录

[语法规则 1](#_Toc39763604)

[消除左递归 3](#_Toc39763605)

[新的语法规则 4](#_Toc39763606)

[抽象语法树 6](#_Toc39763607)

# 语法规则

Goal-> MainClass { ClassDeclaration } EOF

MainClass->**"class"** Identifier **"{"** **"public"** **"static"** **"void"** **"main"** **"("** **"String"** **"["** **"]"** Identifier **")"** **"{"** Statement **"}"** **"}"**

ClassDeclaration->**"class"** Identifier [ **"extends"** Identifier ] **"{"** { VarDeclaration } { MethodDeclaration } **"}"**

VarDeclaration ->Type Identifier **";"**

MethodDeclaration->**"public"** Type Identifier **"("** [ Type Identifier { **","** Type Identifier } ] **")"** **"{"** { VarDeclaration } { Statement } **"return"** Expression **";"** **"}"**

Type->**"int"** **"["** **"]"**

|**"boolean"**

|**"int"**

| Identifier

Statement->**"{"** { Statement } **"}"**

|**"if"** **"("** Expression **")"** Statement **"else"** Statement

|**"while"** **"("** Expression **")"** Statement

|**"System.out.println"** **"("** Expression **")"** **";"**

| Identifier **"="** Expression **";"**

| Identifier **"["** Expression **"]"** **"="** Expression **";"**

Expression-> Expression ( " &&" | "<" | "+" | "-" | "\*" ) Expression

| Expression **"["** Expression **"]"**

| Expression **"."** **"length"**

| Expression **"."** Identifier **"("** [ Expression { **","** Expression } ] **")"**

| IntegerLiteral

|**"true"**

|**"false"**

| Identifier

|**"this"**

|**"new"** **"int"** **"["** Expression **"]"**

|**"new"** Identifier "(" ")"

| **"!"** Expression

|**"("** Expression **")"**

# 消除左递归

由上式可知，只有最后一个表达式有左递归

Expression-> IntegerLiteral L

|**"true"** L

|**"false"** L

| Identifier L

|**"this"** L

|**"new"** **"int"** **"["** Expression **"]"** L

|**"new"** Identifier "(" ")" L

| **"!"** Expression L

|**"("** Expression **")"** L

L-> ( " &&" | "<" | "+" | "-" | "\*" ) Expression L

| **"["** Expression **"]"** L

| **"."** **"length"** L

| **"."** Identifier **"("** [ Expression { **","** Expression } ] **")"** L

| ε

# 新的语法规则

Goal-> MainClass { ClassDeclaration } EOF

MainClass->**"class"** Identifier **"{"** **"public"** **"static"** **"void"** **"main"** **"("** **"String"** **"["** **"]"** Identifier **")"** **"{"** Statement **"}"** **"}"**

ClassDeclaration->**"class"** Identifier [ **"extends"** Identifier ] **"{"** { VarDeclaration } { MethodDeclaration } **"}"**

VarDeclaration ->Type Identifier **";"**

MethodDeclaration->**"public"** Type Identifier **"("** [ Type Identifier { **","** Type Identifier } ] **")"** **"{"** { VarDeclaration } { Statement } **"return"** Expression **";"** **"}"**

Type->**"int"** **"["** **"]"**

|**"boolean"**

|**"int"**

| Identifier

Statement->**"{"** { Statement } **"}"**

|**"if"** **"("** Expression **")"** Statement **"else"** Statement

|**"while"** **"("** Expression **")"** Statement

|**"System.out.println"** **"("** Expression **")"** **";"**

| Identifier **"="** Expression **";"**

| Identifier **"["** Expression **"]"** **"="** Expression **";"**

Expression->IntegerLiteral L

|**"true"** L

|**"false"** L

| Identifier L

|**"this"** L

|**"new"** **"int"** **"["** Expression **"]"** L

|**"new"** Identifier "(" ")" L

| **"!"** Expression L

|**"("** Expression **")"** L

L-> ( " &&" | "<" | "+" | "-" | "\*" ) Expression L

| **"["** Expression **"]"** L

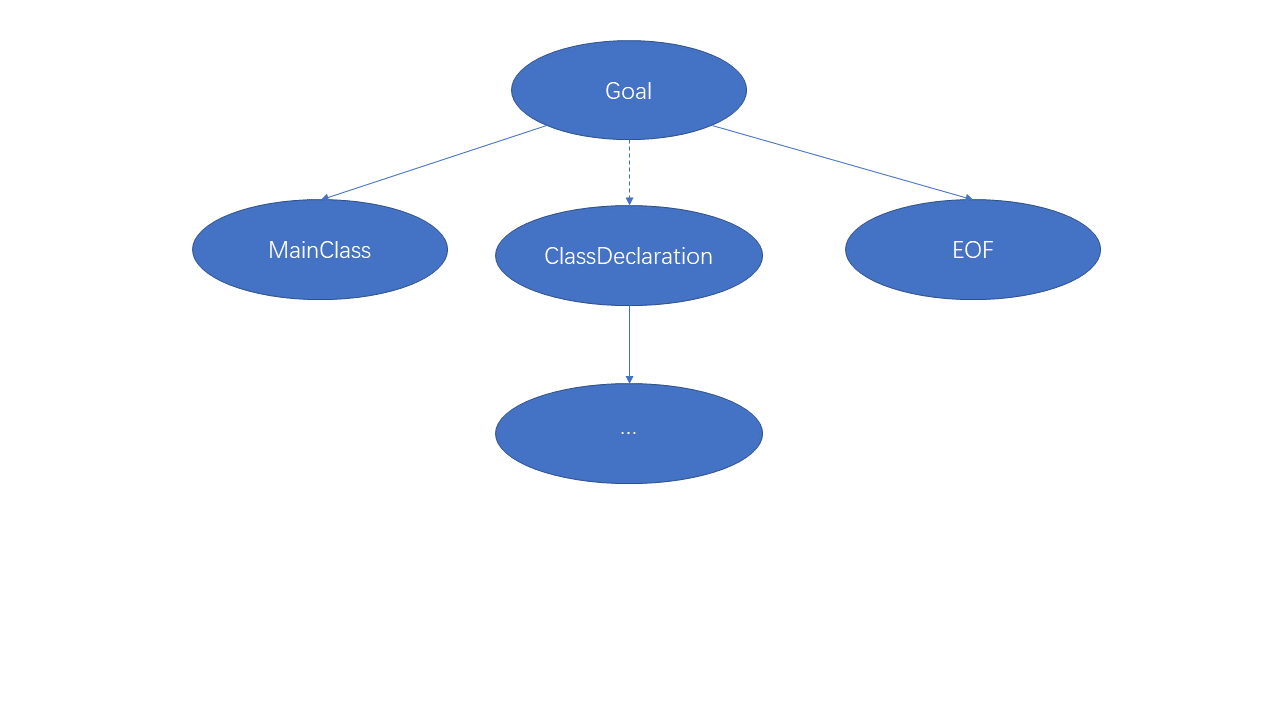
| **"."** **"length"** L

| **"."** Identifier **"("** [ Expression { **","** Expression } ] **")"** L

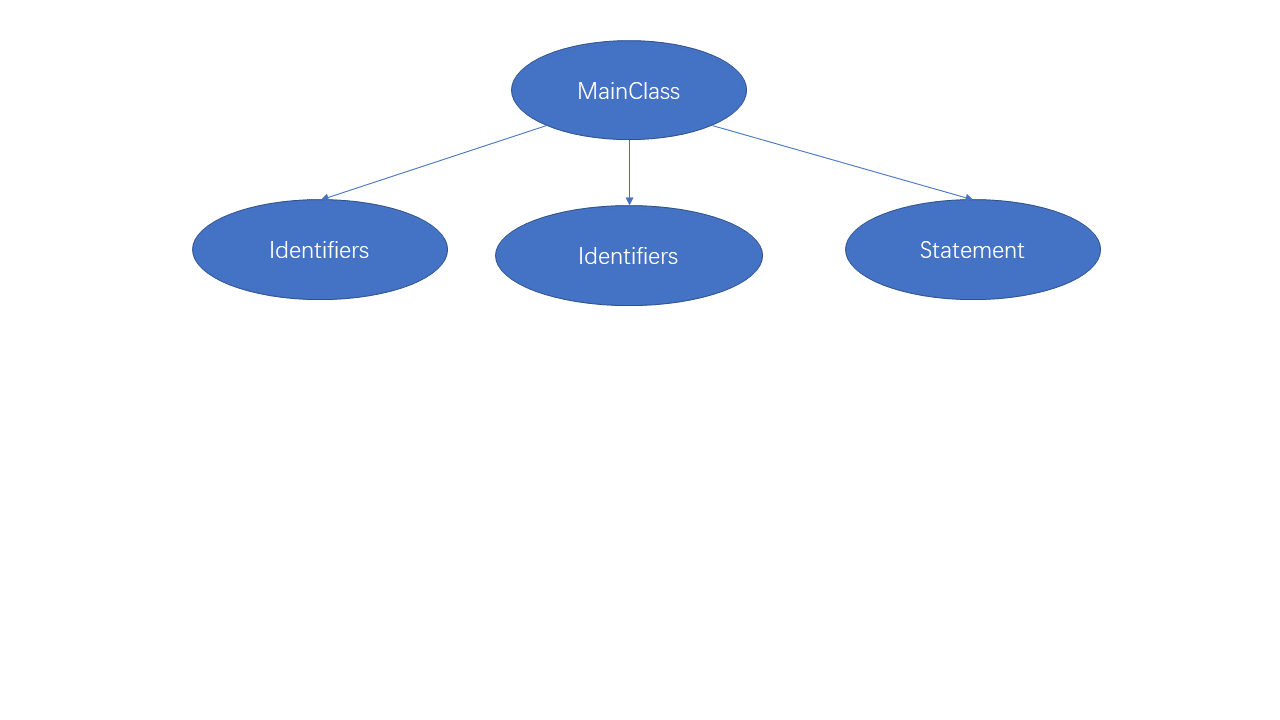
| ε

# 抽象语法树

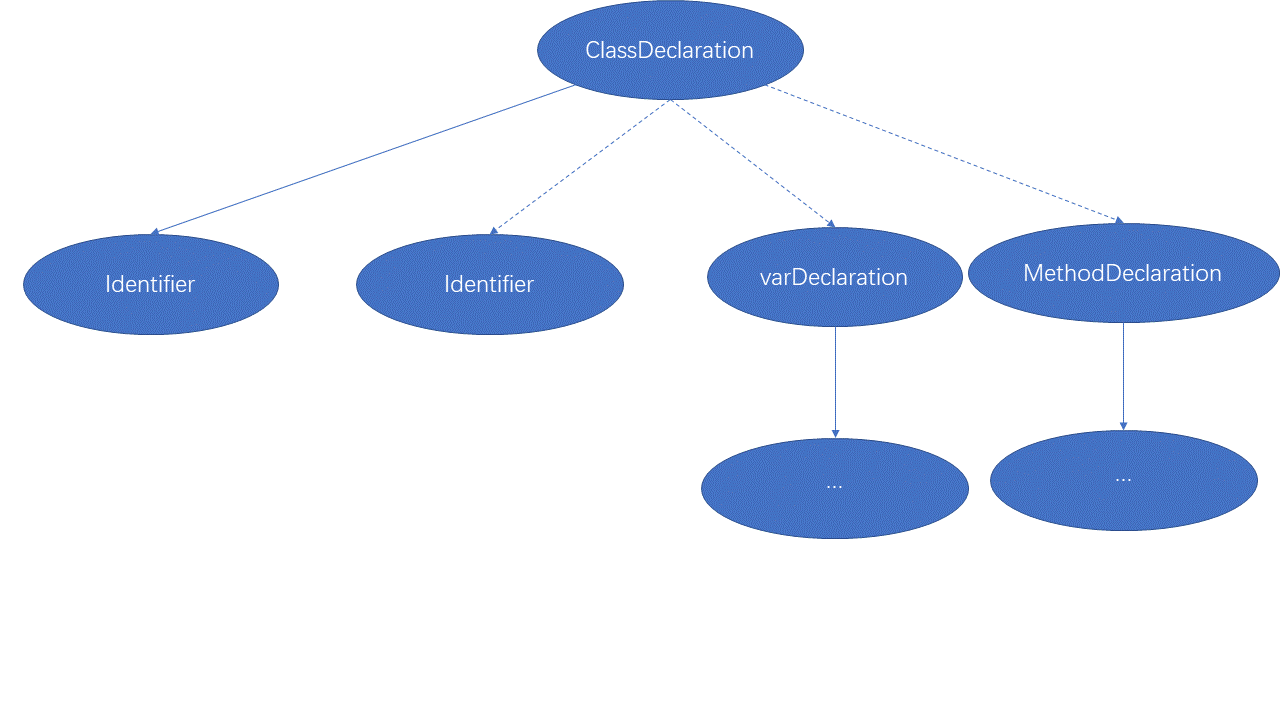
Goal-> MainClass { ClassDeclaration } EOF



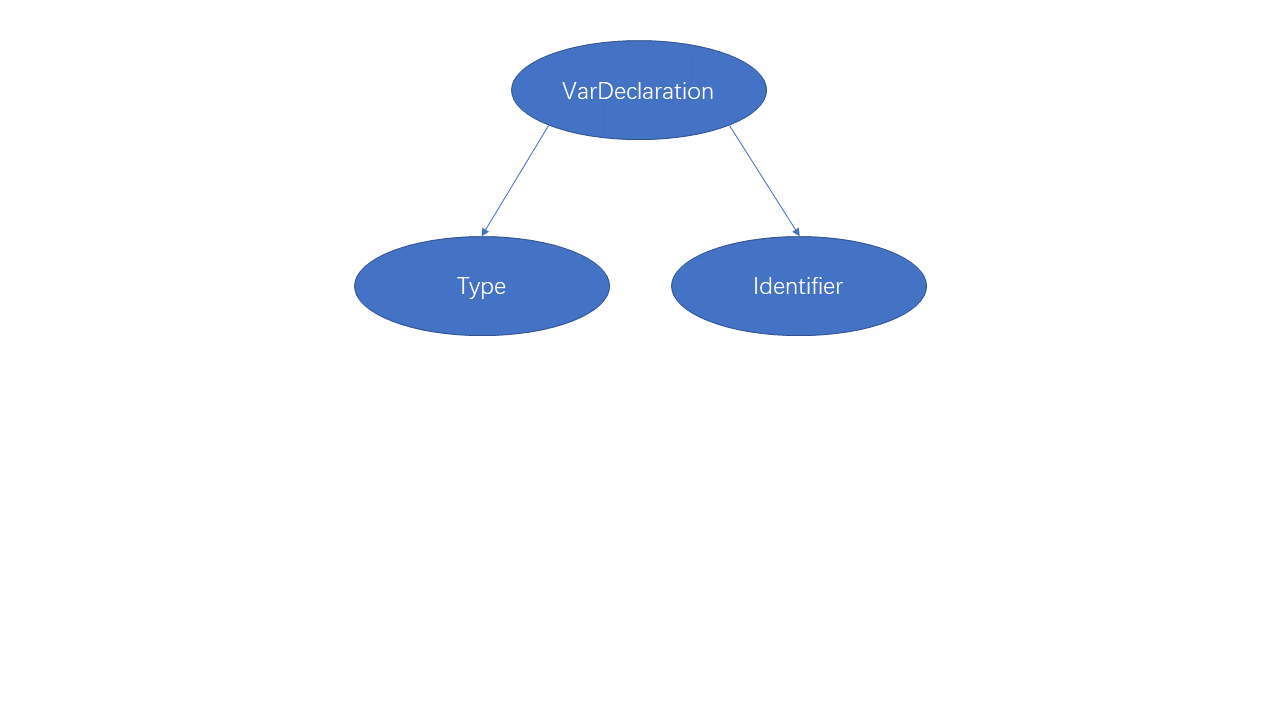
MainClass->**"class"** Identifier **"{"** **"public"** **"static"** **"void"** **"main"** **"("** **"String"** **"["** **"]"** Identifier **")"** **"{"** Statement **"}"** **"}"**



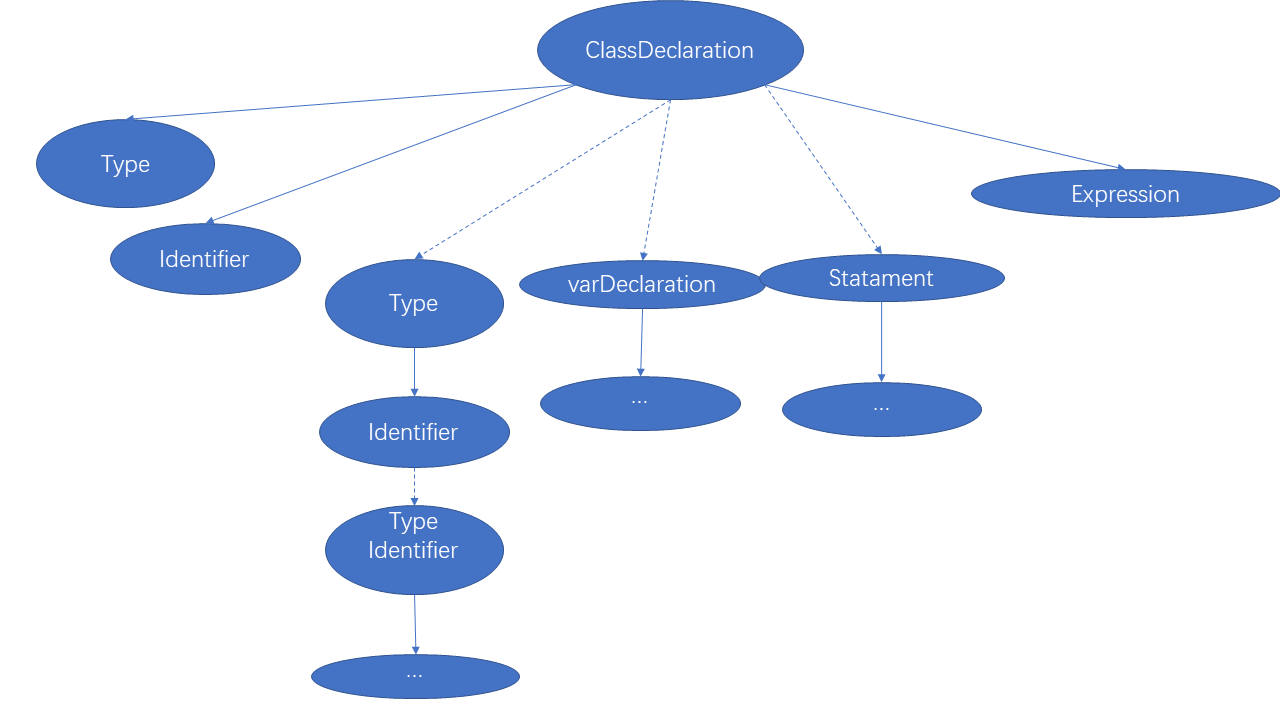
ClassDeclaration->**"class"** Identifier [ **"extends"** Identifier ] **"{"** { VarDeclaration } { MethodDeclaration } **"}"**



VarDeclaration ->Type Identifier **";"**



MethodDeclaration->**"public"** Type Identifier **"("** [ Type Identifier { **","** Type Identifier } ] **")"** **"{"** { VarDeclaration } { Statement } **"return"** Expression **";"** **"}"**

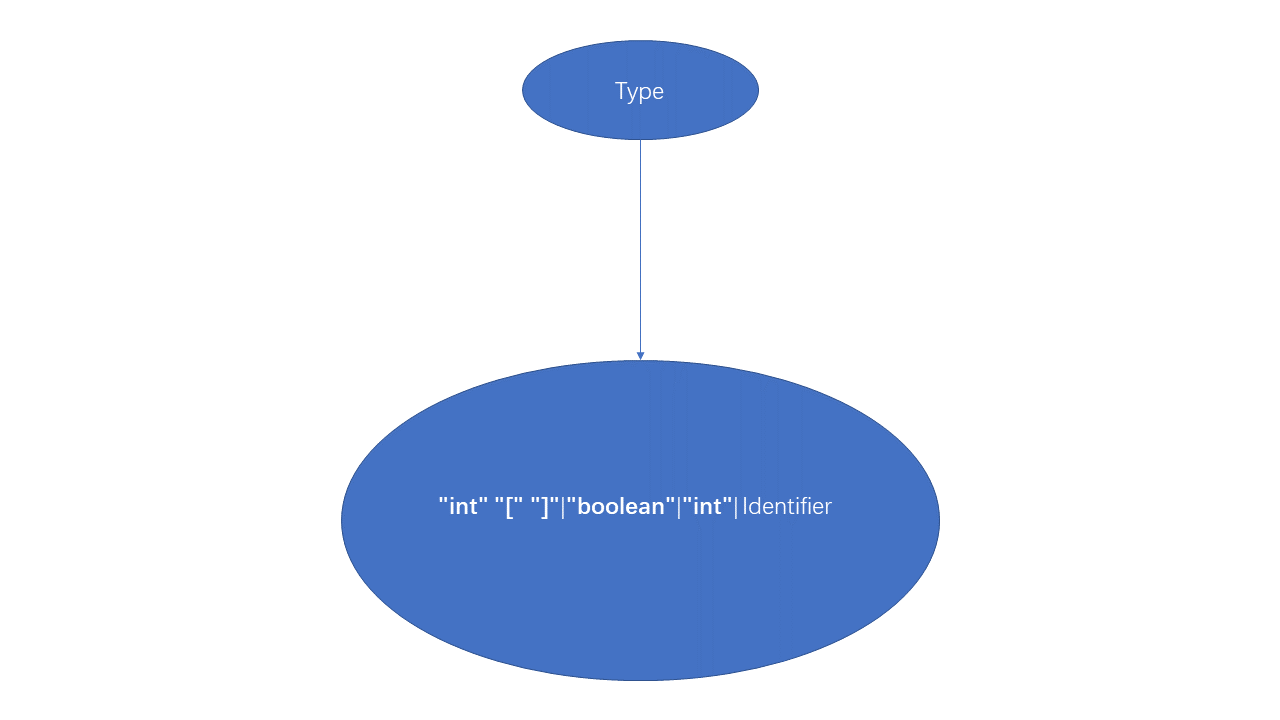


Type->**"int"** **"["** **"]"**

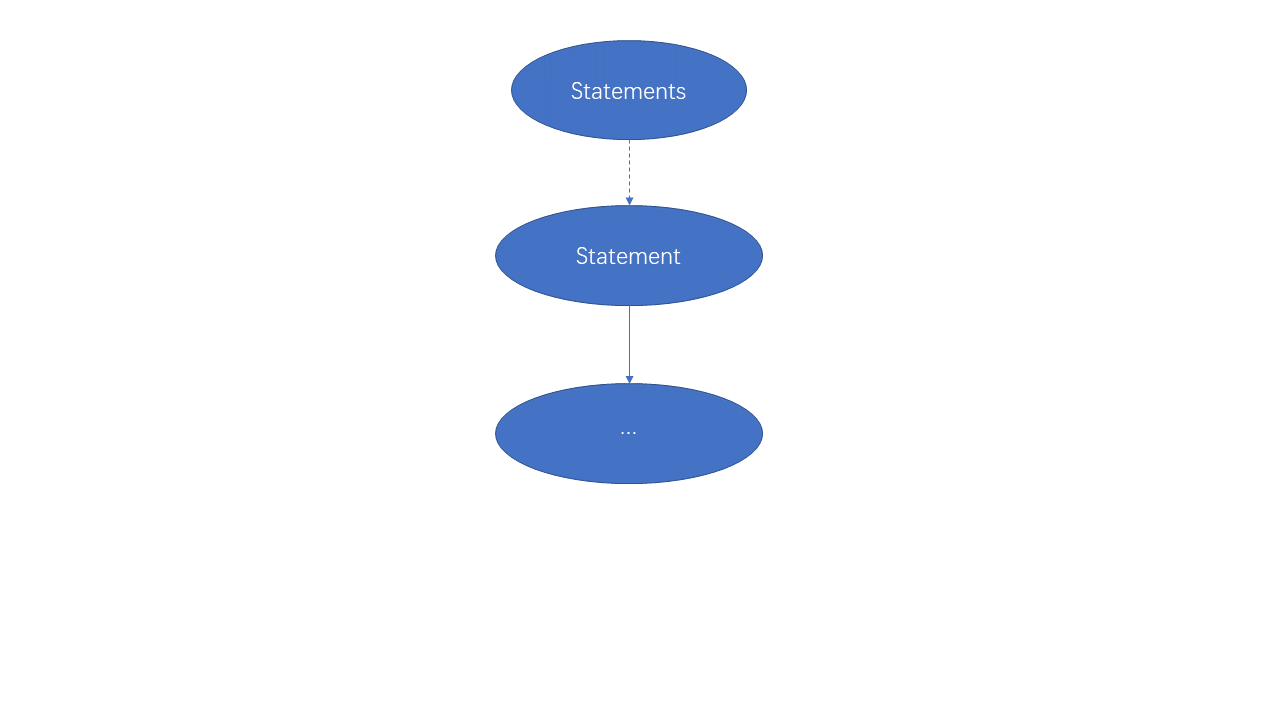
|**"boolean"**

|**"int"**

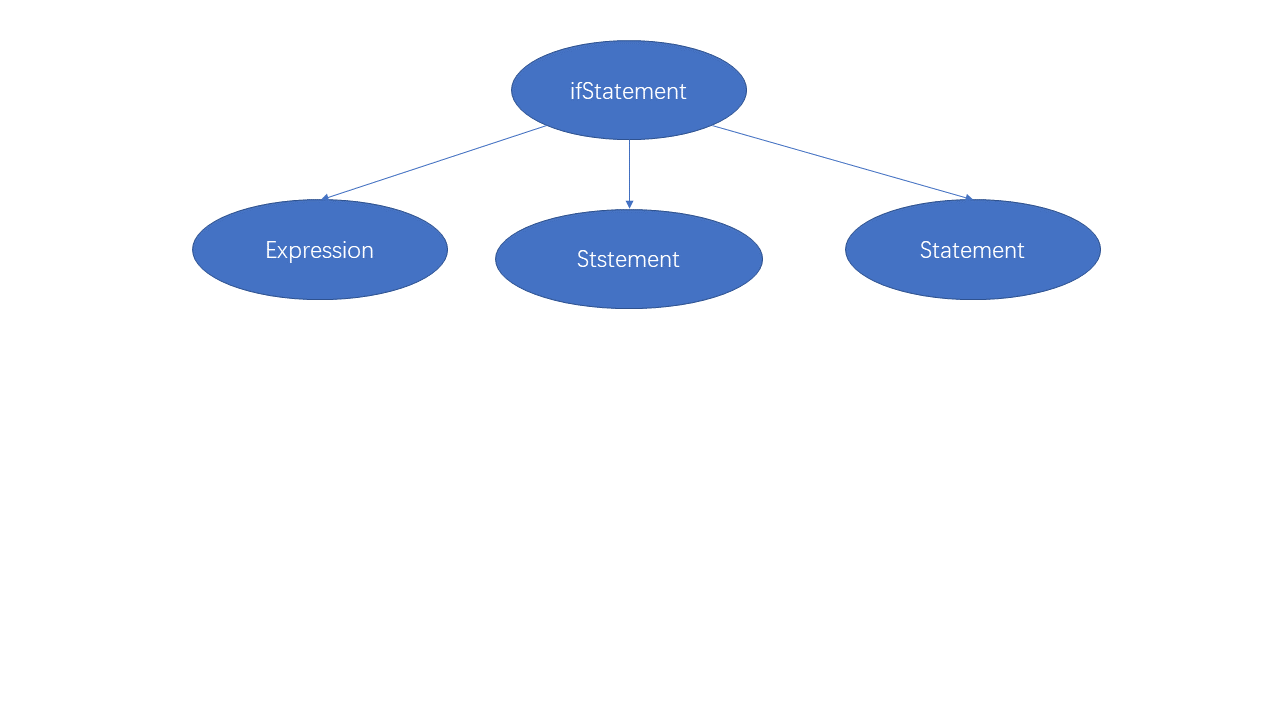
| Identifier



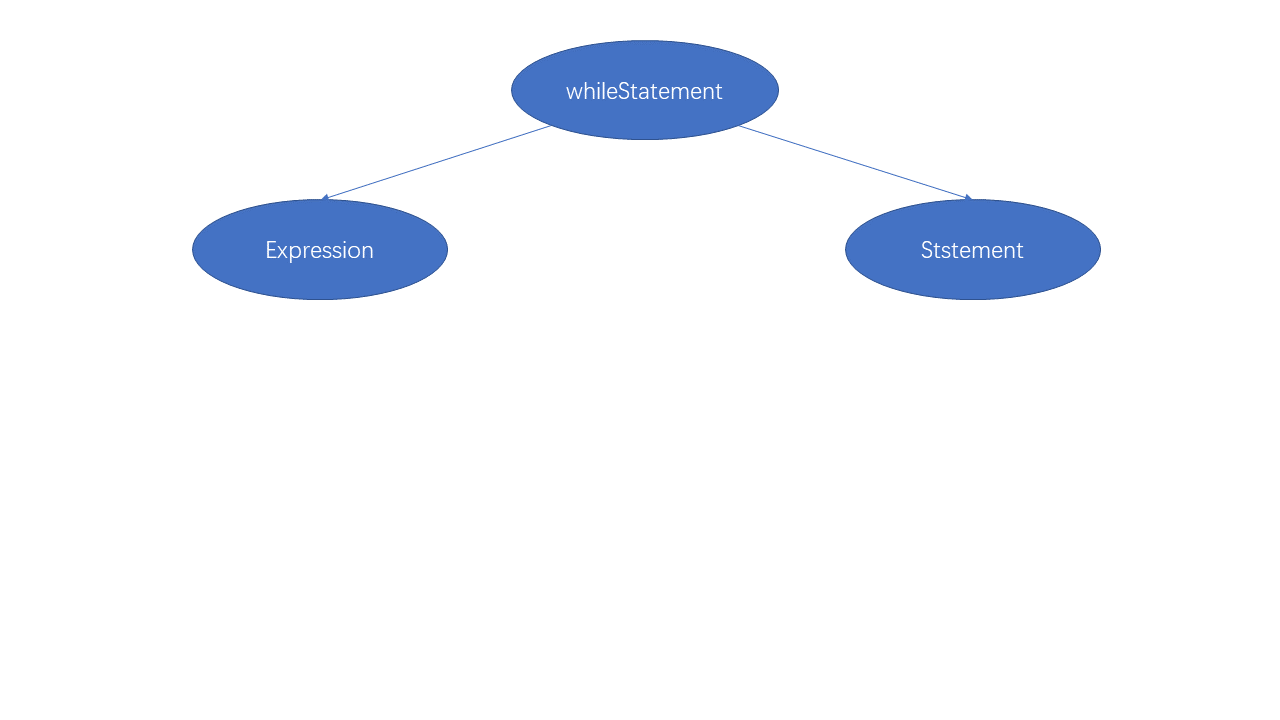
Statements->**"{"** { Statement } **"}"**



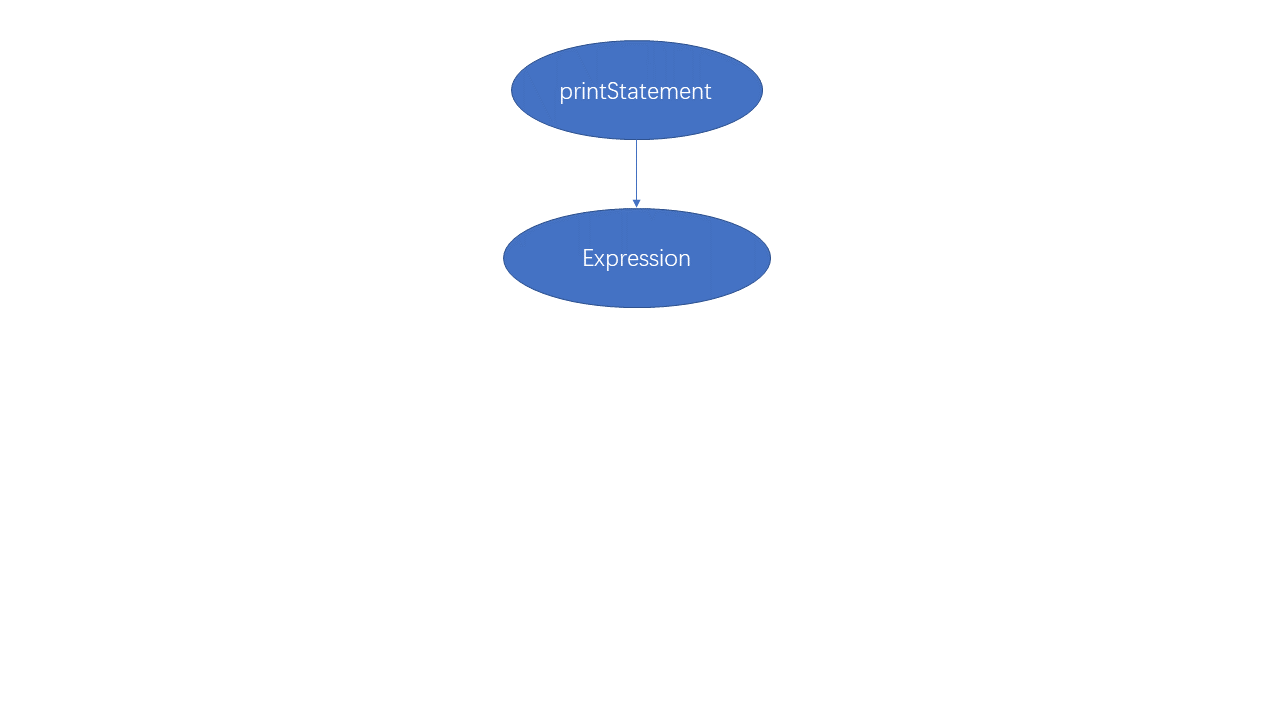
IfStatement ->**"if"** **"("** Expression **")"** Statement **"else"** Statement



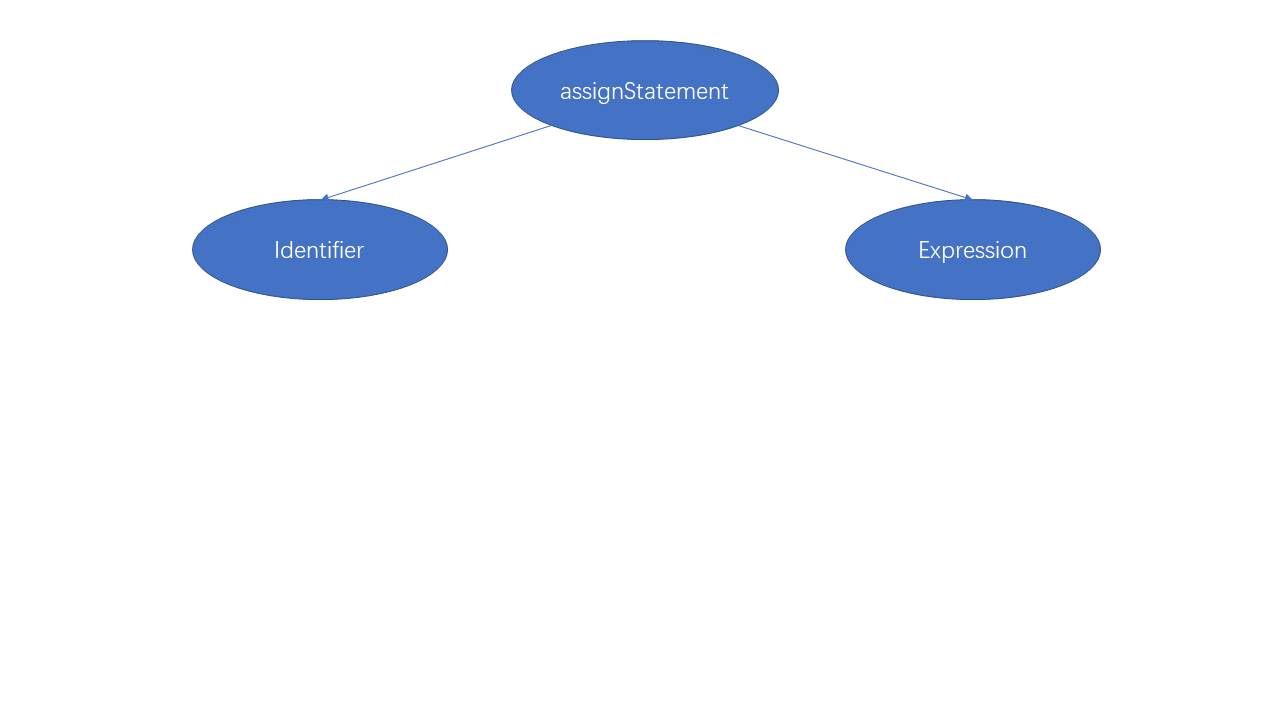
WhileStatement ->**"while"** **"("** Expression **")"** Statement



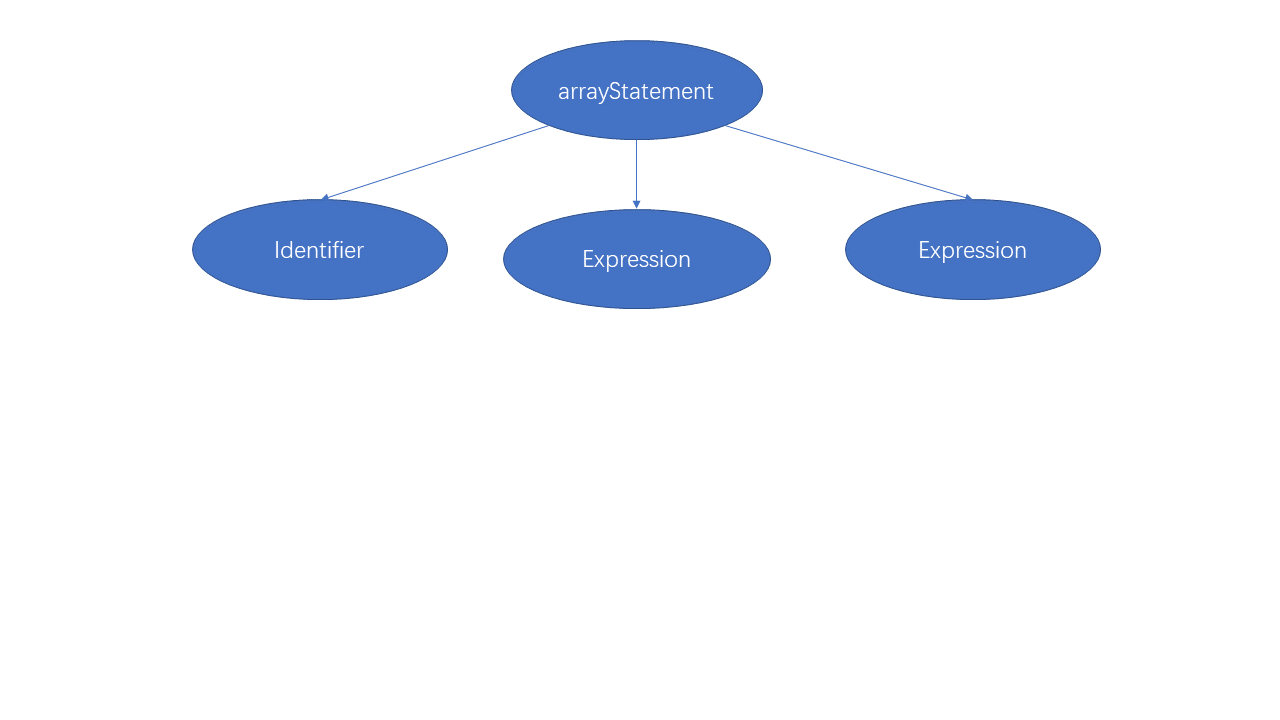
PrintStatement ->**"System.out.println"** **"("** Expression **")"** **";"**



AssignStatement -> Identifier **"="** Expression **";"**



ArrayStatement -> Identifier **"["** Expression **"]"** **"="** Expression **";"**



Statement-> Statements

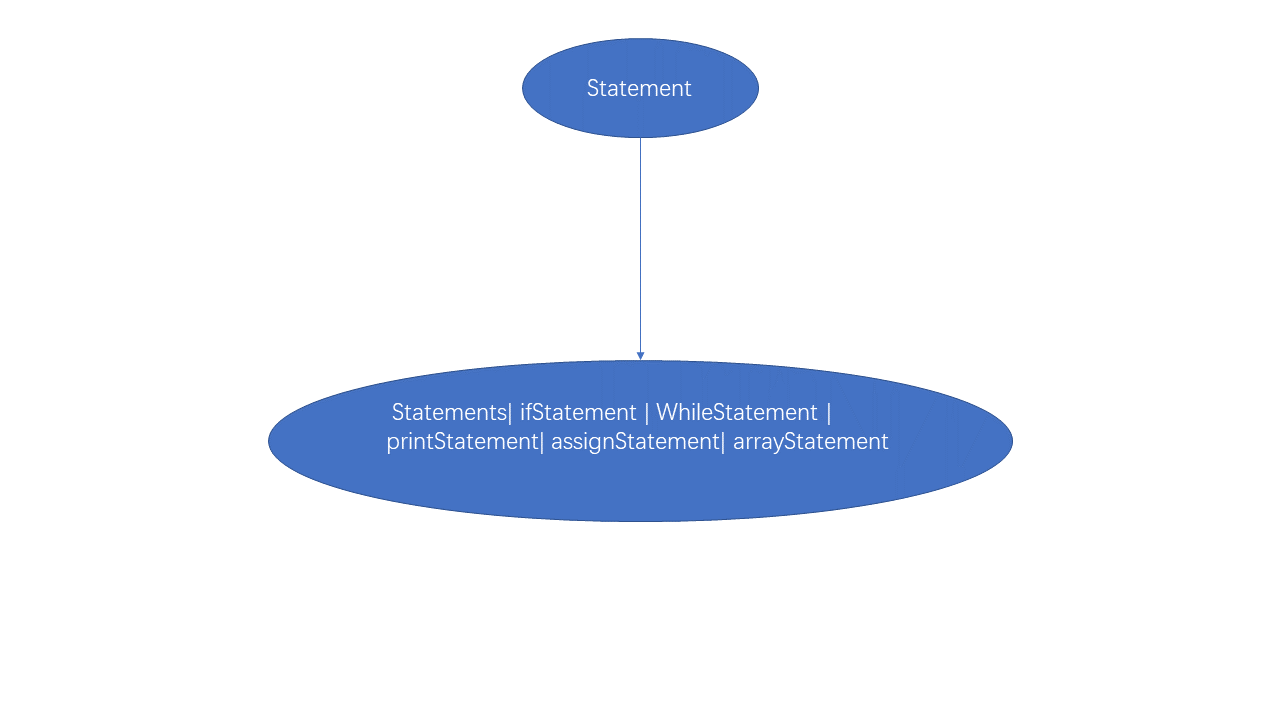
| IfStatement

| WhileStatement

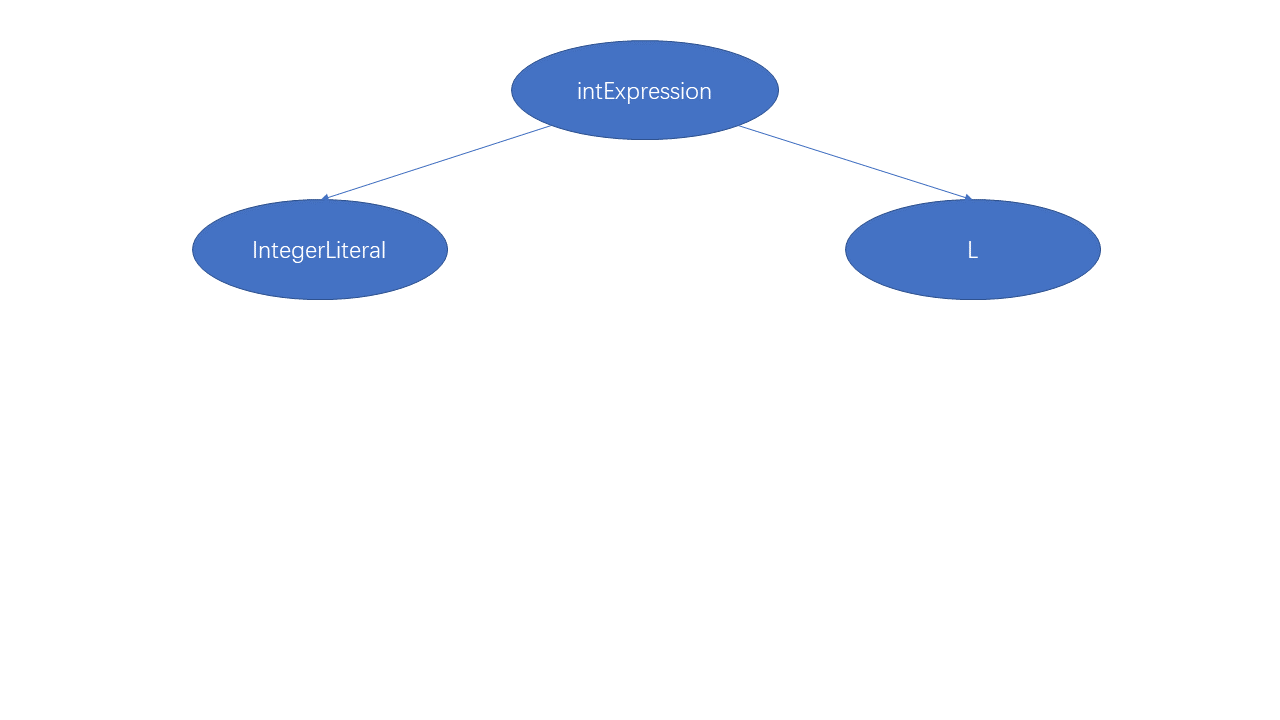
| PrintStatement

| AssignStatement

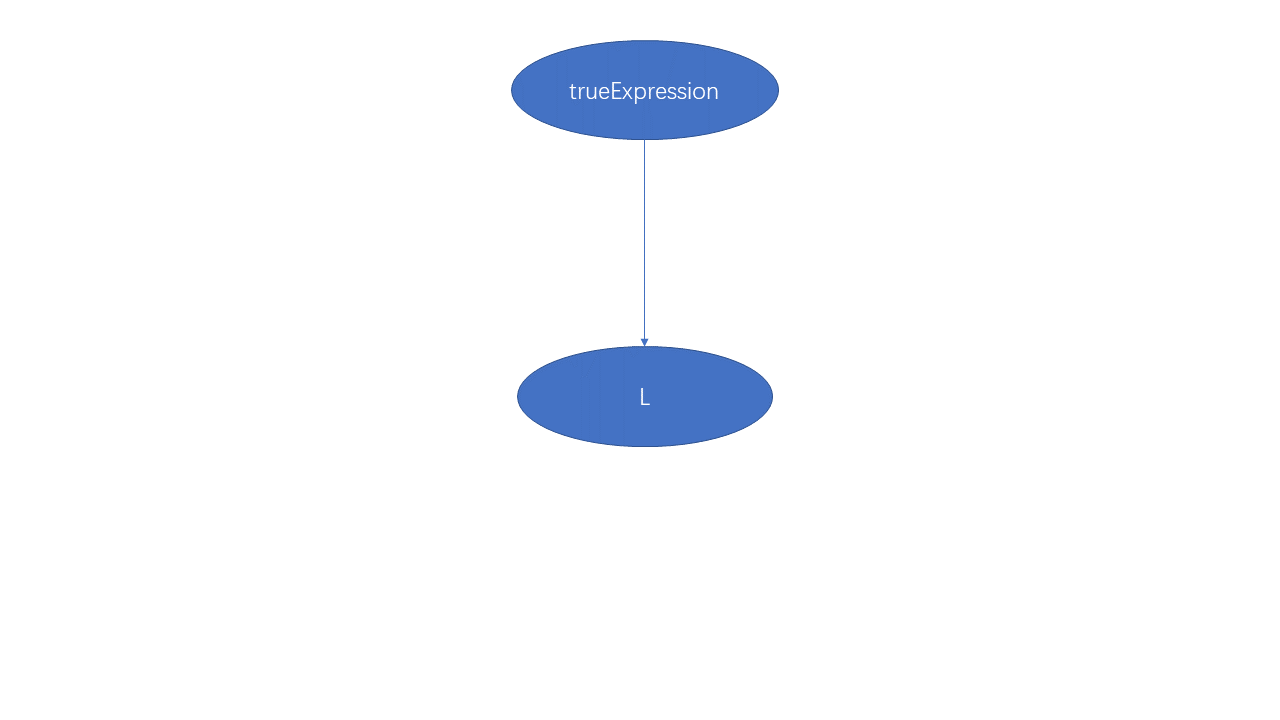
| ArrayStatement



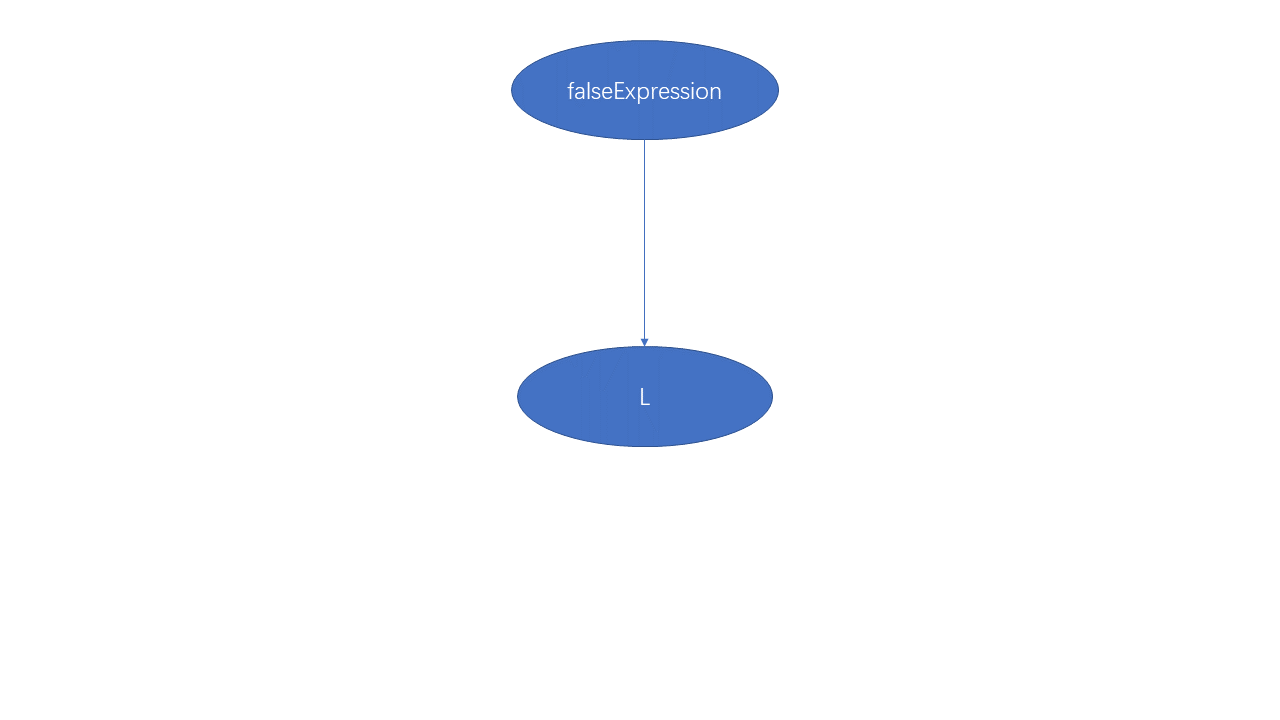
IntExpression-> IntegerLiteral L



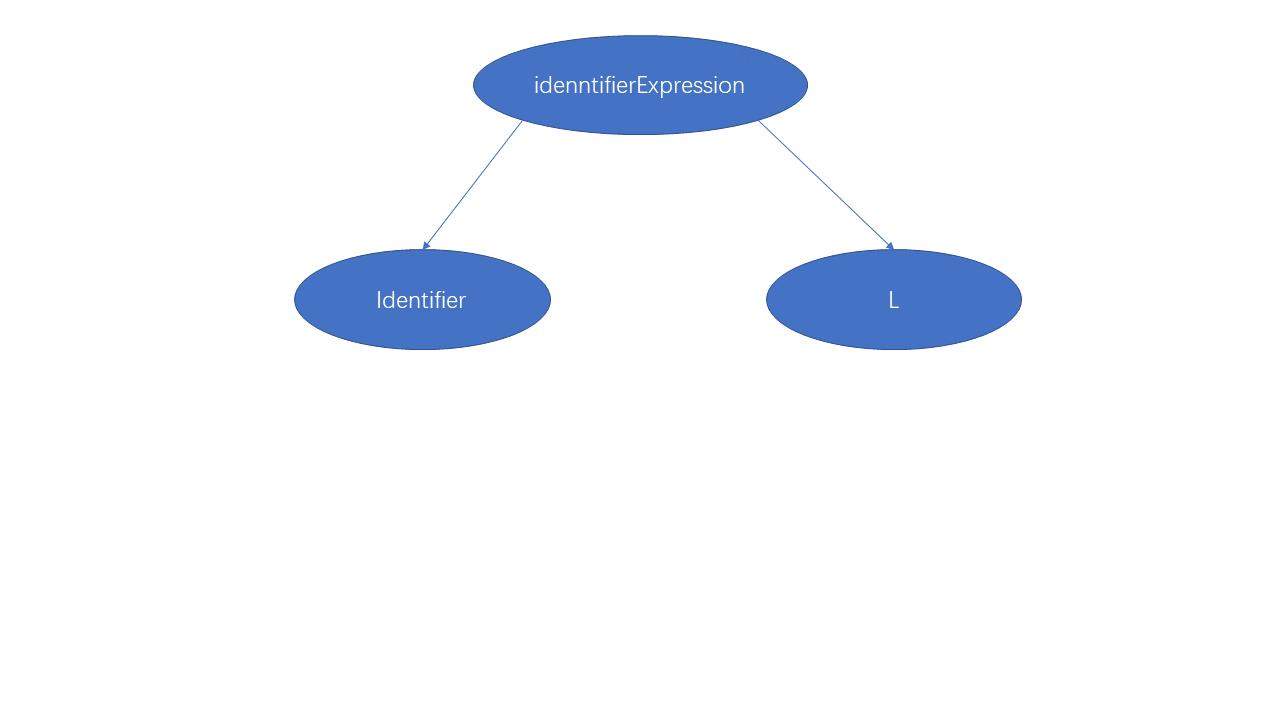
TrueExpression->**"true"** L



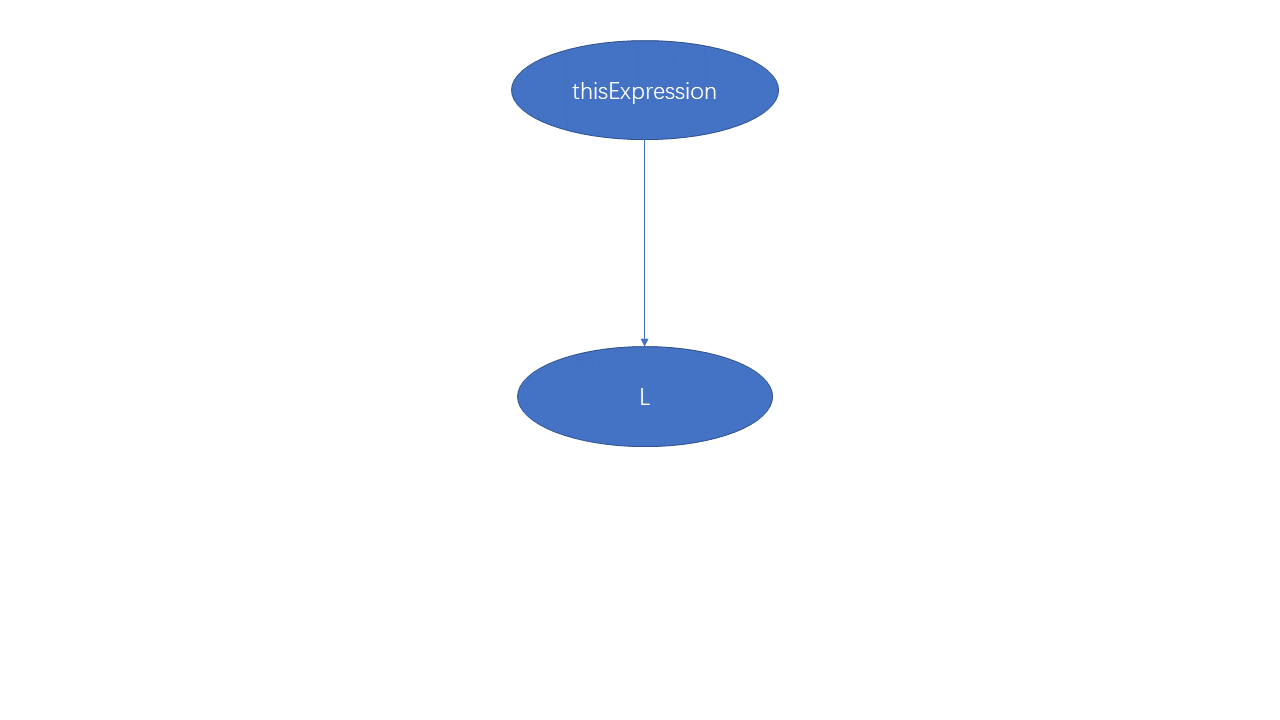
FalseExpression->**"false"** L



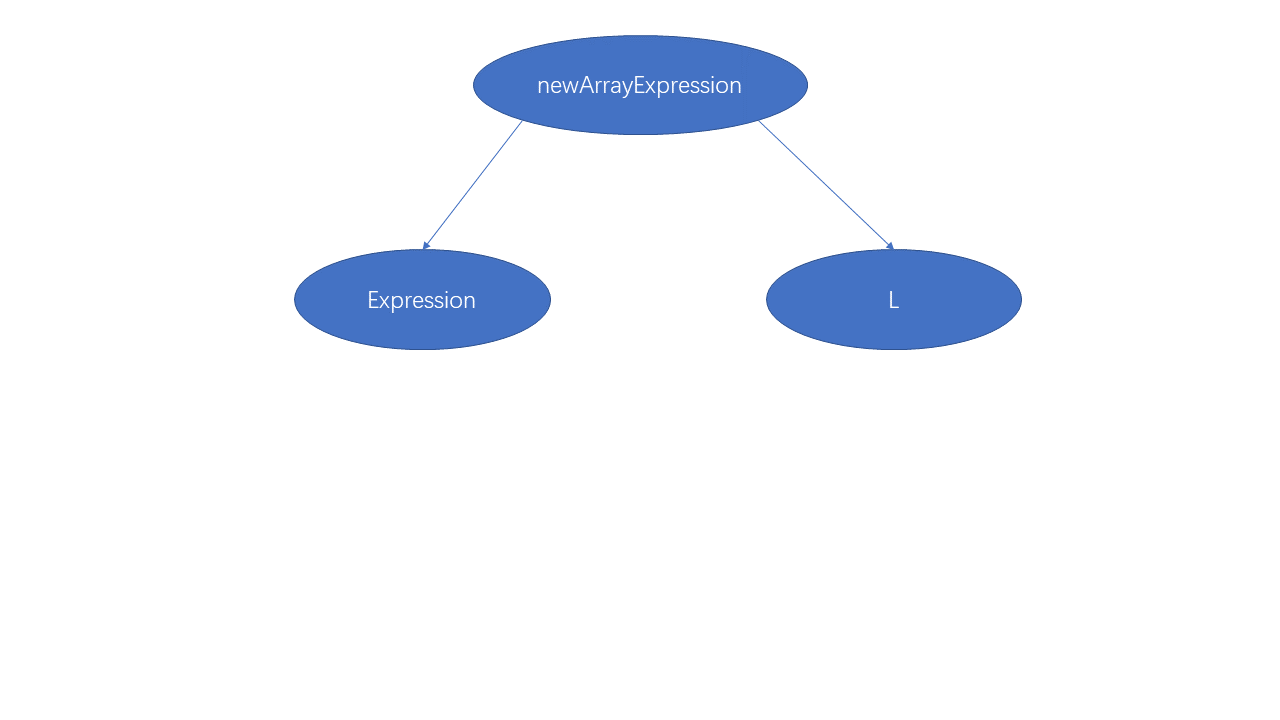
IdentifierExpression-> Identifier L



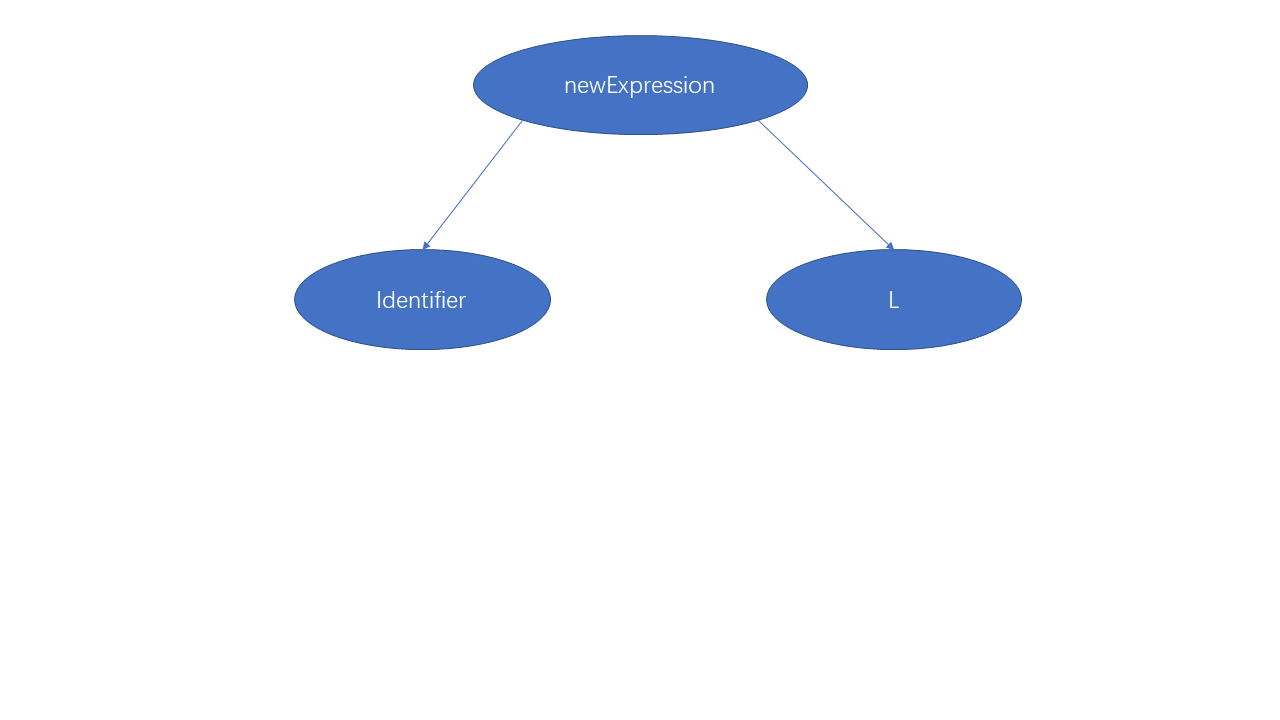
ThisExpression->**"this"** L



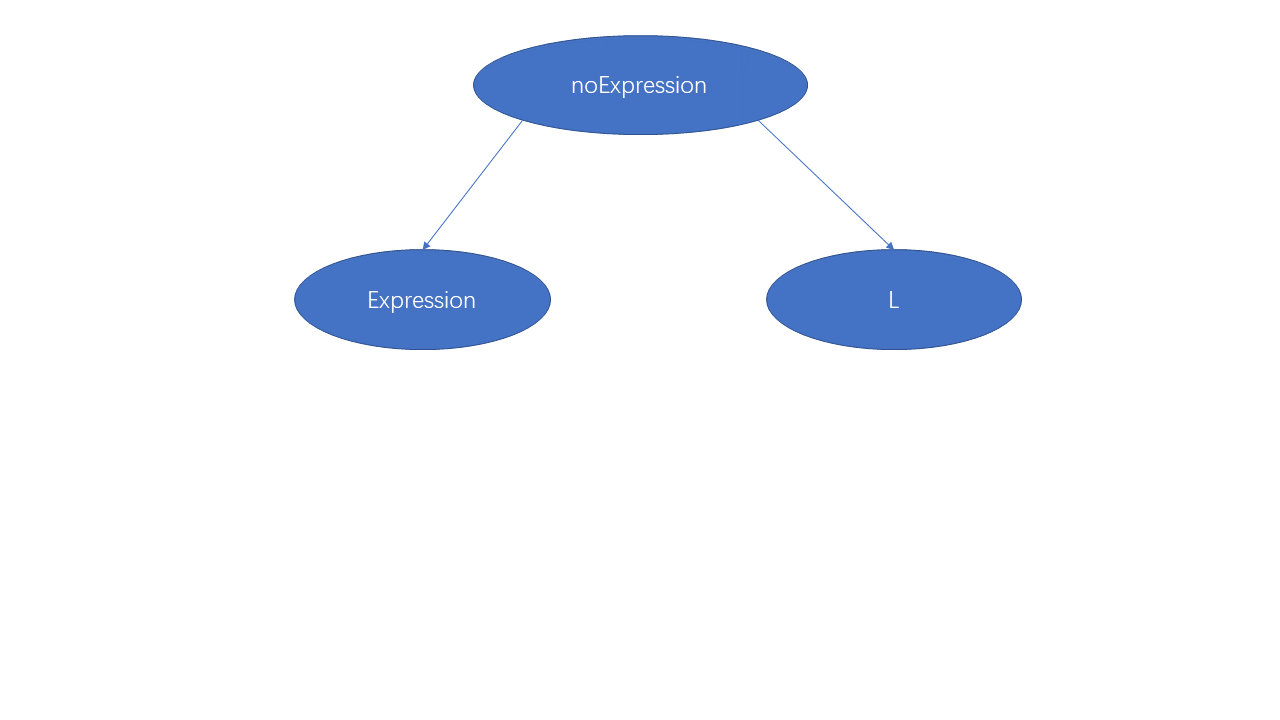
NewArrayExpression->**"new"** **"int"** **"["** Expression **"]"** L



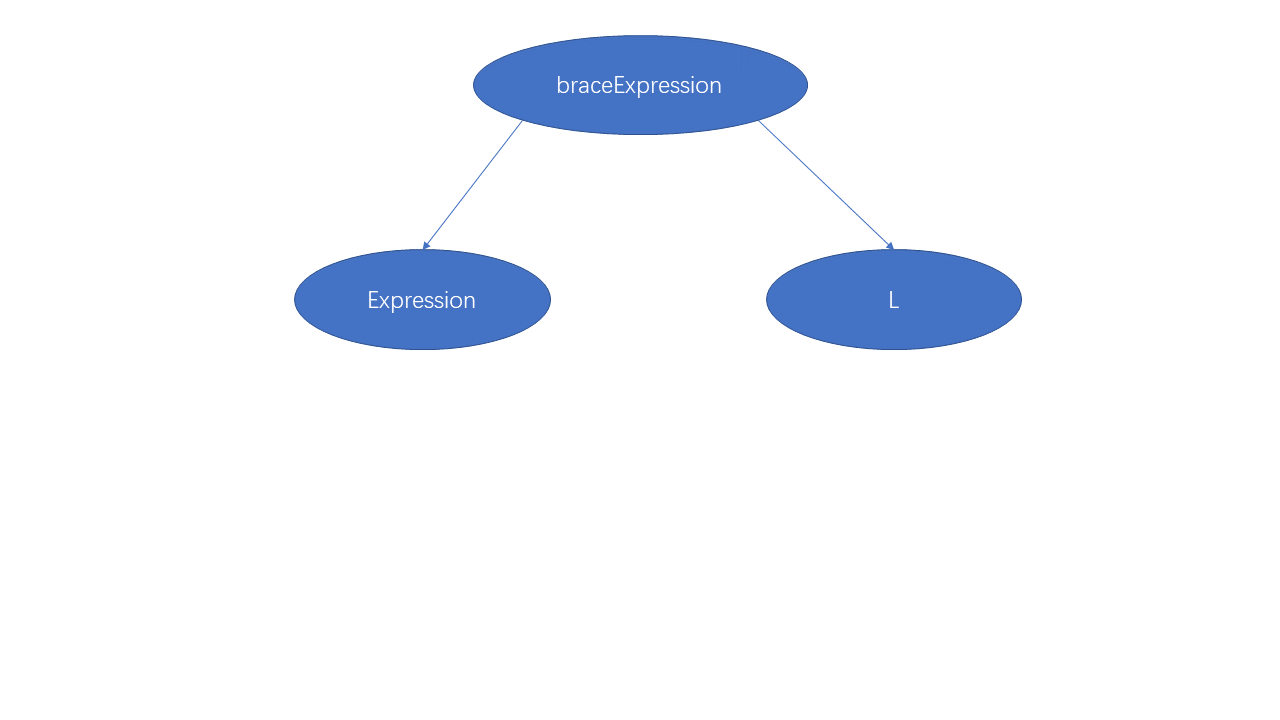
NewExpression->**"new"** Identifier "(" ")" L



NoExpression-> **"!"** Expression L



BraceExpression->**"("** Expression **")"** L



Expression-> IntExpression

| TrueExpression

| FalseExpression

|IdentifierExpression

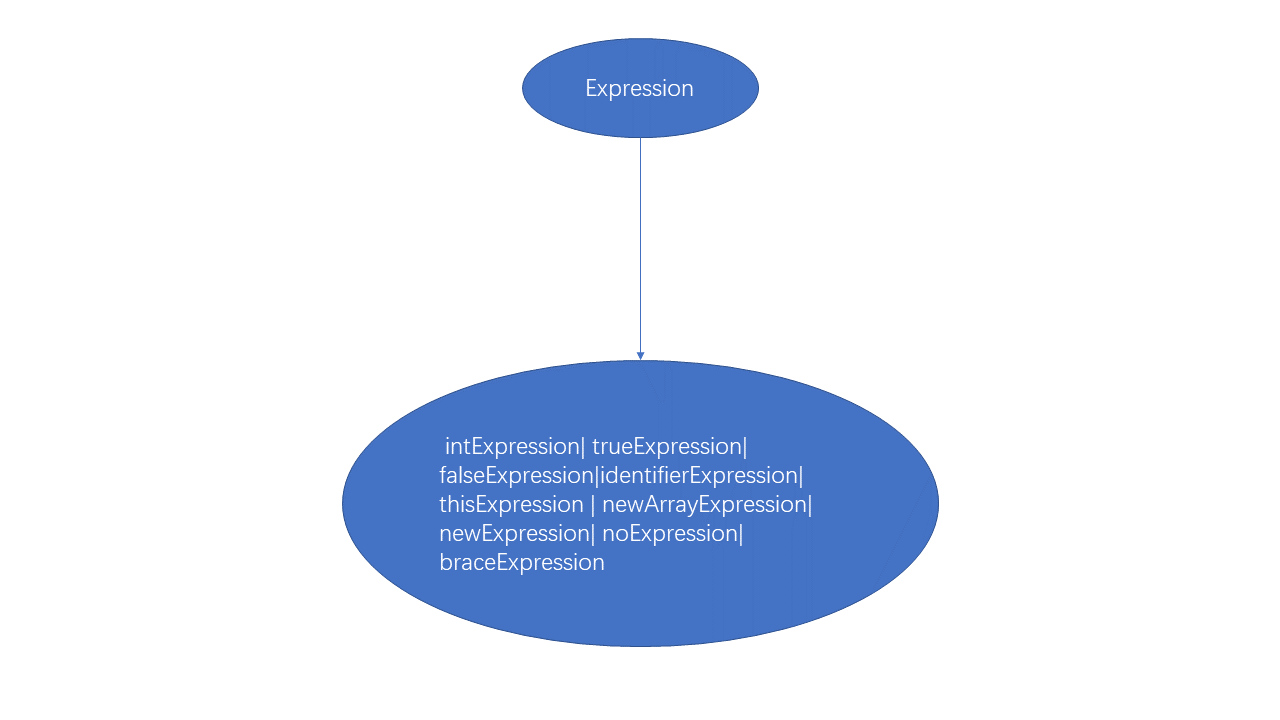
| ThisExpression

| NewArrayExpression

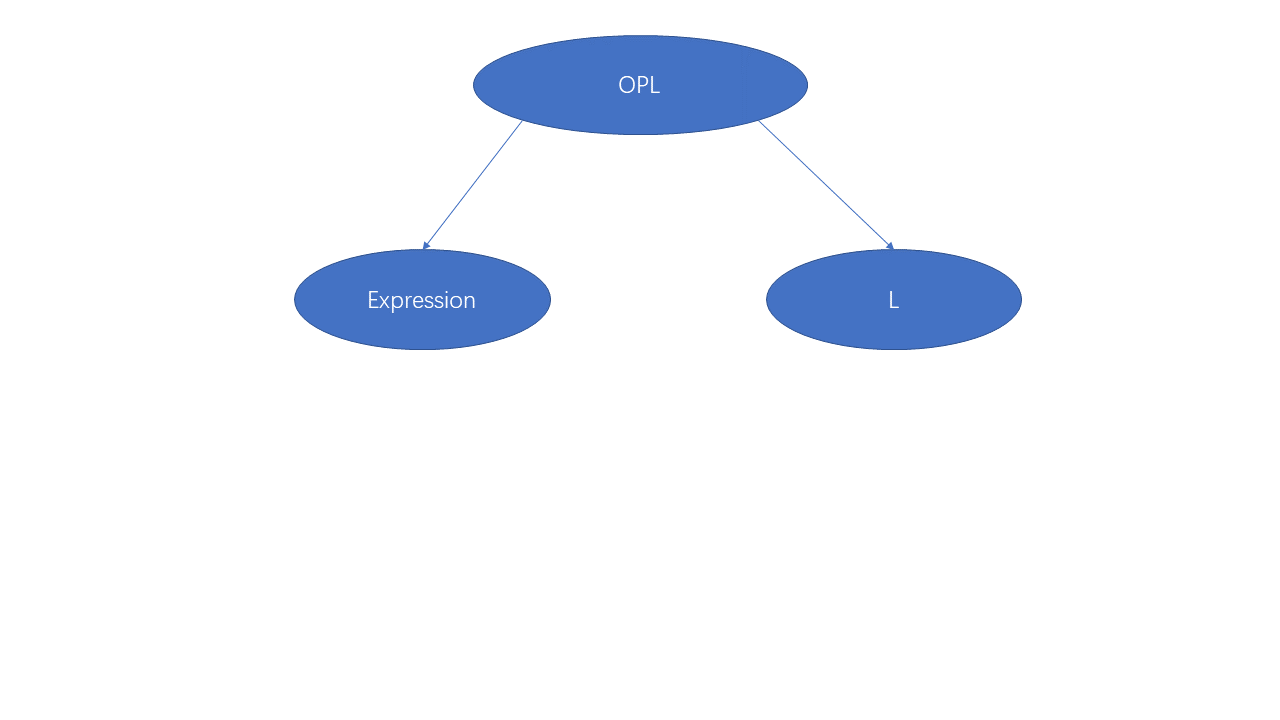
| NewExpression

| NoExpression

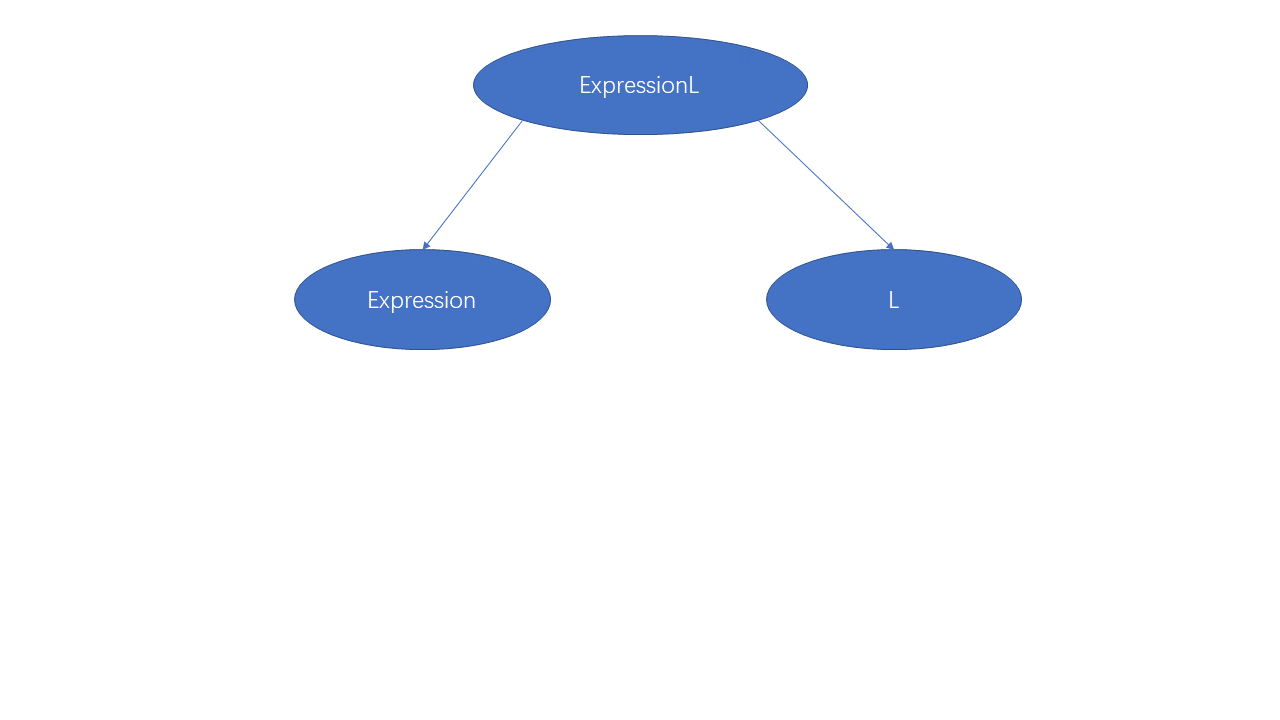
| BraceExpression



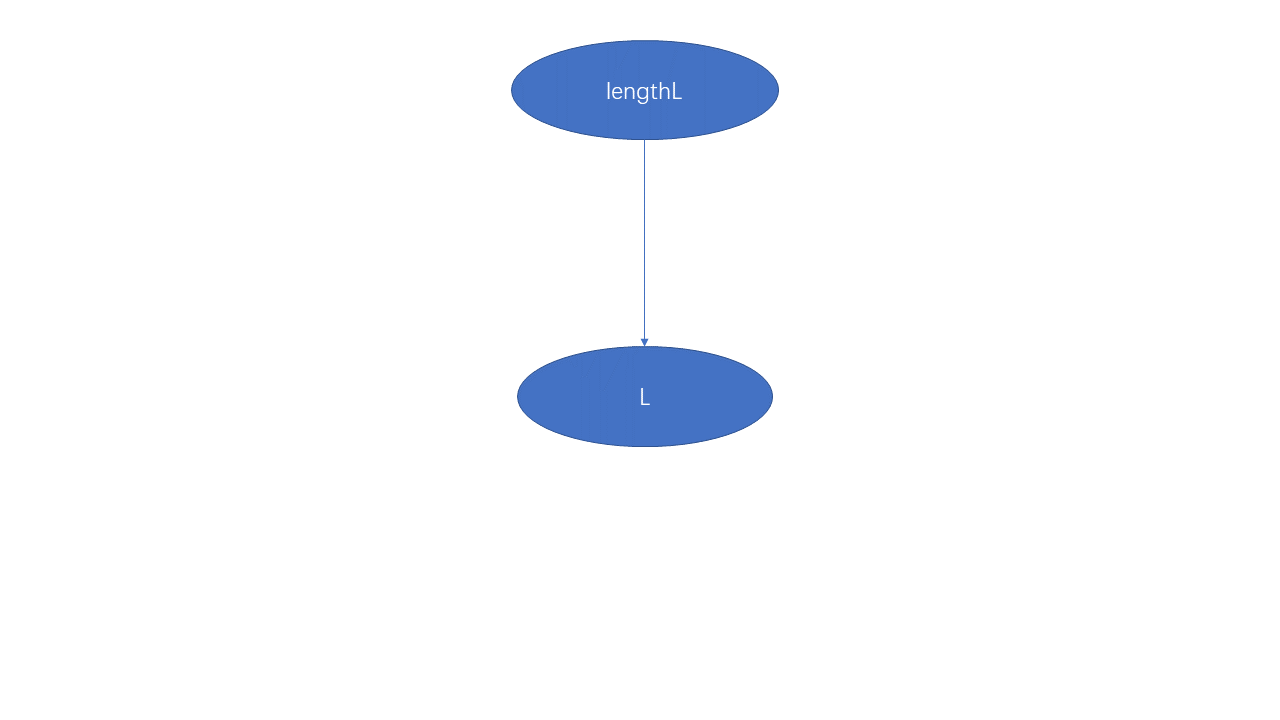
OPL->( " &&" | "<" | "+" | "-" | "\*" ) Expression L



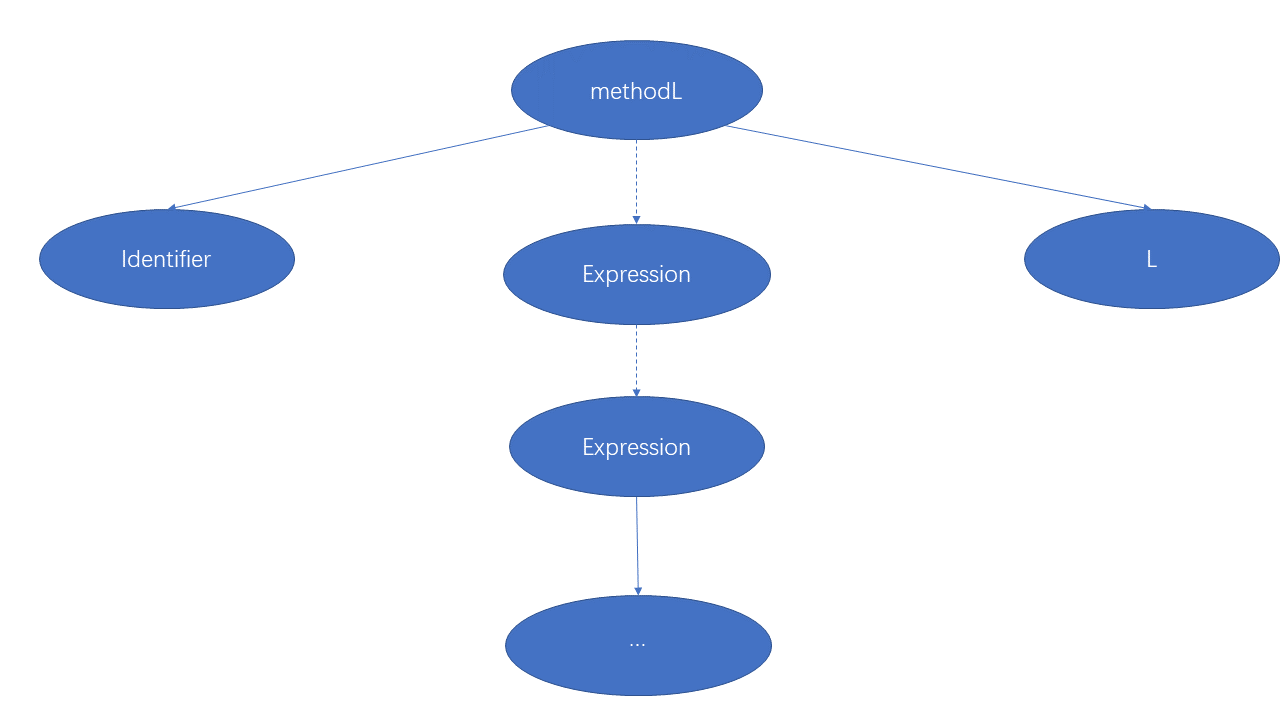
ExpressionL-> **"["** Expression **"]"** L



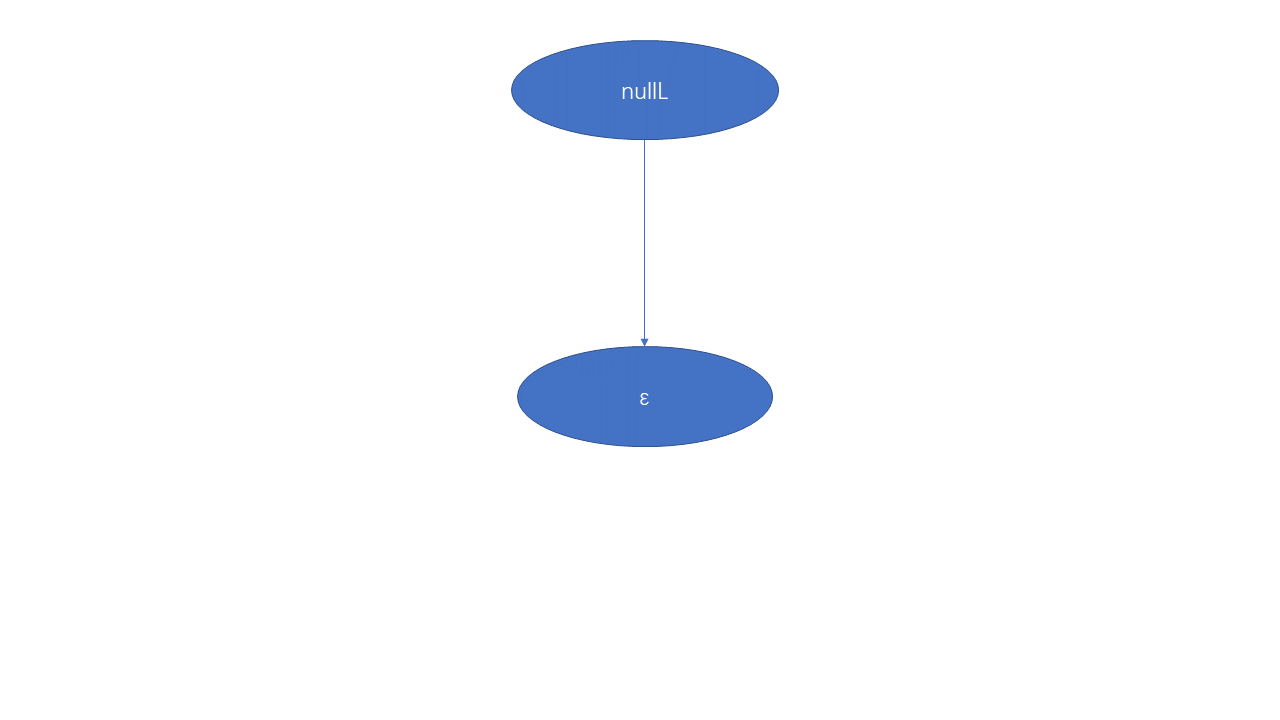
LengthL->**"."** **"length"** L



MethodL->**"."** Identifier **"("** [ Expression { **","** Expression } ] **")"** L



nullL->ε



L-> OPL

|ExpressionL

|LengthL

| MethodL

| NullL

