

```

/**
 * Kotlin syntax grammar in ANTLR4 notation
 */

// SECTION: general

kotlinFile
    : packageHeader importList topLevelObject*
    ;

packageHeader
    : (PACKAGE identifier)?
    ;

importList
    : importHeader*
    ;

importHeader
    : IMPORT identifier MULT?
    ;

topLevelObject
    : declaration
    ;

declaration
    : classDeclaration
    | functionDeclaration
    | propertyDeclaration
    ;

// SECTION: classes

classDeclaration
    : modifiers? (CLASS | FUN? INTERFACE) simpleIdentifier
    typeParameters? primaryConstructor?
    delegationSpecifiers?
    (classBody | enumClassBody)?
    ;

primaryConstructor
    : (modifiers? CONSTRUCTOR)? classParameters
    ;

classBody

```

```

        : classMemberDeclarations
        ;

classParameters
    : classParameter*
    ;

classParameter
    : modifiers? VAL? simpleIdentifier type expression?
    ;

delegationSpecifiers
    : annotatedDelegationSpecifier*
    ;

delegationSpecifier
    : constructorInvocation
    | userType
    | functionType
    ;

constructorInvocation
    : userType valueArguments
    ;

annotatedDelegationSpecifier
    : annotation* delegationSpecifier
    ;

typeParameters
    : LANGLE typeParameter+ RANGLE
    ;

typeParameter
    : simpleIdentifier type?
    ;

// SECTION: classMembers

classMemberDeclarations
    : classMemberDeclaration*
    ;

classMemberDeclaration
    : declaration
    ;

```

```
functionValueParameters
    : functionValueParameter*
    ;
```

```
functionValueParameter
    : parameter expression?
    ;
```

```
functionDeclaration
    : modifiers?
    FUN typeParameters? simpleIdentifier
    functionValueParameters
    type? functionBody?
    ;
```

```
functionBody
    : block
    | expression
    ;
```

```
variableDeclaration
    : annotation* simpleIdentifier type?
    ;
```

```
propertyDeclaration
    : modifiers? VAL typeParameters? variableDeclaration expression?
    ;
```

```
parameter
    : simpleIdentifier type
    ;
```

## // SECTION: enumClasses

```
enumClassBody
    : enumEntries? classMemberDeclarations?
    ;
```

```
enumEntries
    : enumEntry*
    ;
```

```
enumEntry
    : modifiers? simpleIdentifier valueArguments? classBody?
    ;
```

## // SECTION: types

```
type
  : parenthesizedType
  | typeReference
  | functionType
  ;

typeReference
  : userType
  | DYNAMIC
  ;

userType
  : simpleUserType+
  ;

simpleUserType
  : simpleIdentifier (typeArguments)?
  ;

typeProjection
  : type | MULT
  ;

functionType
  : functionTypeParameters type
  ;

functionTypeParameters
  : (parameter | type)*
  ;

parenthesizedType
  : type
  ;
```

## // SECTION: statements

```
statements
  : statement*
  ;

statement
  : annotation*
```

```

    ( declaration
    | loopStatement
    | expression)
    ;

controlStructureBody
    : block
    | statement
    ;

block
    : statements
    ;

loopStatement
    : forStatement
    | whileStatement
    | doWhileStatement
    ;

forStatement
    : annotation* variableDeclaration expression
    controlStructureBody?
    ;

whileStatement
    : expression controlStructureBody?
    ;

doWhileStatement
    : controlStructureBody? expression
    ;

// SECTION: expressions

expression
    : disjunction
    ;

disjunction
    : conjunction+
    ;

conjunction
    : equality+
    ;

```

```

equality
    : comparison (equalityOperator comparison)*
    ;

comparison
    : infixOperation (comparisonOperator infixOperation)?
    ;

infixOperation
    : elvisExpression (inOperator elvisExpression)*
    ;

elvisExpression
    : infixFunctionCall
    ;

infixFunctionCall
    : rangeExpression (simpleIdentifier rangeExpression)*
    ;

rangeExpression
    : additiveExpression+
    ;

additiveExpression
    : multiplicativeExpression (additiveOperator
multiplicativeExpression)*
    ;

multiplicativeExpression
    : asExpression (multiplicativeOperator asExpression)*
    ;

asExpression
    : comparisonWithLiteralRightSide
    ;

comparisonWithLiteralRightSide
    : prefixUnaryExpression (LANGE literalConstant RANGLE
expression)*
    ;

prefixUnaryExpression
    : unaryPrefix* postfixUnaryExpression
    ;

```

```

unaryPrefix
  : annotation
  | prefixUnaryOperator
  ;

postfixUnaryExpression
  : primaryExpression postfixUnarySuffix*
  ;

postfixUnarySuffix
  | typeArguments
  | callSuffix
  | indexingSuffix
  | navigationSuffix
  ;

indexingSuffix
  : expression+
  ;

navigationSuffix
  : memberAccessOperator (simpleIdentifier |
  parenthesizedExpression | CLASS)
  ;

callSuffix
  : typeArguments? valueArguments? annotatedLambda
  | typeArguments? valueArguments
  ;

annotatedLambda
  : annotation* lambdaLiteral
  ;

typeArguments
  : LANGLE typeProjection+ RANGLE
  ;

valueArguments
  : valueArgument*
  ;

valueArgument
  : annotation? simpleIdentifier? MULT? expression
  ;

```

```
primaryExpression
  : parenthesizedExpression
  | simpleIdentifier
  | literalConstant
  | stringLiteral
  | functionLiteral
  | thisExpression
  | superExpression
  | ifExpression
  | whenExpression
  | jumpExpression
  ;
```

```
parenthesizedExpression
  : expression
  ;
```

```
literalConstant
  : BooleanLiteral
  | IntegerLiteral
  | HexLiteral
  | BinLiteral
  | NullLiteral
  ;
```

```
stringLiteral
  : lineStringLiteral
  ;
```

```
lineStringLiteral
  : (lineStringContent | lineStringExpression)*
  ;
```

```
lineStringContent
  : LineStrText
  | LineStrEscapedChar
  | LineStrRef
  ;
```

```
lineStringExpression
  : expression
  ;
```

```
lambdaLiteral
  : lambdaParameters? statements
```



```

;

lambdaParameters
  : lambdaParameter+
  ;

lambdaParameter
  : variableDeclaration
  ;

functionLiteral
  : lambdaLiteral
  ;

thisExpression
  : THIS
  ;

superExpression
  : SUPER (LANGLE type RANGLE)? (AT_NO_WS simpleIdentifier)?
  ;

ifExpression
  : expression controlStructureBody?
  | expression controlStructureBody? ELSE controlStructureBody?
  ;

whenSubject
  : (annotation* VAL variableDeclaration)? expression
  ;

whenExpression
  : whenSubject? whenEntry*
  ;

whenEntry
  : whenCondition+ controlStructureBody
  | ELSE controlStructureBody
  ;

whenCondition
  : expression
  ;

jumpExpression
  : RETURN expression?

```

```

        | CONTINUE
        | BREAK
        ;

equalityOperator
    : EXCL_EQ
    | EQEQ
    ;

comparisonOperator
    : LANGLE
    | RANGLE
    | LE
    | GE
    ;

inOperator
    ;

additiveOperator
    : ADD | SUB
    ;

multiplicativeOperator
    : MULT
    ;

prefixUnaryOperator
    : SUB
    | ADD
    | excl
    ;

excl
    : EXCL_NO_WS
    | EXCL_WS
    ;

memberAccessOperator
    ;

// SECTION: modifiers

modifiers
    : (annotation | modifier)+
    ;

```

```
modifier
  : classModifier
  | memberModifier
  | propertyModifier
  | inheritanceModifier
  ;
```

```
classModifier
  : ENUM
  | SEALED
  | ANNOTATION
  | DATA
  | INNER
  ;
```

```
memberModifier
  : OVERRIDE
  | LATEINIT
  ;
```

```
propertyModifier
  : CONST
  ;
```

```
inheritanceModifier
  : ABSTRACT
  | FINAL
  | OPEN
  ;
```

## // SECTION: annotations

```
annotation
  : singleAnnotation
  ;
```

```
singleAnnotation
  : (AT_NO_WS | AT_PRE_WS) unescapedAnnotation
  ;
```

```
unescapedAnnotation
  : constructorInvocation
  | userType
  ;
```

## // SECTION: identifiers

simpleIdentifier: Identifier

ABSTRACT  
ANNOTATION  
BY  
CATCH  
COMPANION  
CONSTRUCTOR  
CROSSINLINE  
DATA  
DYNAMIC  
ENUM  
EXTERNAL  
FINAL  
FINALLY  
GET  
IMPORT  
INFIX  
INIT  
INLINE  
INNER  
INTERNAL  
LATEINIT  
NOINLINE  
OPEN  
OPERATOR  
OUT  
OVERRIDE  
PRIVATE  
PROTECTED  
PUBLIC  
REIFIED  
SEALED  
TAILREC  
SET  
VARARG  
WHERE  
FIELD  
PROPERTY  
RECEIVER  
PARAM  
SETPARAM  
DELEGATE  
FILE  
EXPECT  
ACTUAL

```
| CONST  
| SUSPEND  
;
```

```
identifier  
    : simpleIdentifier+  
    ;
```

```
/**  
 * Kotlin lexical grammar in ANTLR4 notation  
 */
```

```
// SECTION: separatorsAndOperations
```

```
MULT: '*';  
ADD: '+';  
SUB: '-';  
EXCL_WS: '!' Hidden;  
EXCL_NO_WS: '!';  
AT_NO_WS: '@';  
AT_PRE_WS: (Hidden | NL) '@' ;  
LANGLE: '<';  
RANGLE: '>';  
LE: '<=';  
GE: '>=';  
EXCL_EQ: '!=';  
EQEQ: '==';
```

```
// SECTION: keywords
```

```
FILE: 'file';  
FIELD: 'field';  
PROPERTY: 'property';  
GET: 'get';  
SET: 'set';  
RECEIVER: 'receiver';  
PARAM: 'param';  
SETPARAM: 'setparam';  
DELEGATE: 'delegate';  
  
PACKAGE: 'package';  
IMPORT: 'import';  
CLASS: 'class';  
INTERFACE: 'interface';  
FUN: 'fun';  
VAL: 'val';
```

```
CONSTRUCTOR: 'constructor';
BY: 'by';
COMPANION: 'companion';
INIT: 'init';
THIS: 'this';
SUPER: 'super';
WHERE: 'where';
ELSE: 'else';
CATCH: 'catch';
FINALLY: 'finally';
RETURN: 'return';
CONTINUE: 'continue';
BREAK: 'break';
OUT: 'out';
DYNAMIC: 'dynamic';
```

## **// SECTION: lexicalModifiers**

```
PUBLIC: 'public';
PRIVATE: 'private';
PROTECTED: 'protected';
INTERNAL: 'internal';
ENUM: 'enum';
SEALED: 'sealed';
ANNOTATION: 'annotation';
DATA: 'data';
INNER: 'inner';
TAILREC: 'tailrec';
OPERATOR: 'operator';
INLINE: 'inline';
INFIX: 'infix';
EXTERNAL: 'external';
SUSPEND: 'suspend';
OVERRIDE: 'override';
ABSTRACT: 'abstract';
FINAL: 'final';
OPEN: 'open';
CONST: 'const';
LATEINIT: 'lateinit';
VARARG: 'vararg';
NOINLINE: 'noinline';
CROSSINLINE: 'crossinline';
REIFIED: 'reified';
EXPECT: 'expect';
ACTUAL: 'actual';
```

## // SECTION: literals

```
IntegerLiteral
  : DecDigitNoZero DecDigitOrSeparator* DecDigit
  | DecDigit
  ;
```

```
HexLiteral
  : '0' [xX] HexDigit HexDigitOrSeparator* HexDigit
  | '0' [xX] HexDigit
  ;
```

```
BinLiteral
  : '0' [bB] BinDigit BinDigitOrSeparator* BinDigit
  | '0' [bB] BinDigit
  ;
```

```
BooleanLiteral: 'true'| 'false';
```

```
NullLiteral: 'null';
```

## // SECTION: lexicalIdentifiers

```
Identifier
  : (Letter | '_' ) (Letter | '_' | UnicodeDigit)*
  | '\'' ~([\r\n] | '\'' )+ '\''
  ;
```

## // SECTION: strings

```
LineStrRef
  : FieldIdentifier
  ;
```

```
LineStrText
  : ~('\\" | "'" | '$')+ | '$'
  ;
```

```
LineStrEscapedChar
  : EscapedIdentifier
  | UniCharacterLiteral
  ;
```