The Grammar of Fusion

Anderson Boettge Pinheiro[a], Francisco Heron de Carvalho Junior[a]

[a]*Mestrado e Doutorado em Ciência da Computação*
Universidade Federal do Ceará
{ab.pinheiro,heron,carneiro}@lia.ufc.br

Abstract

This document presents a formal syntax specification for the Fusion language, aimed at extending Java with GPU programming abstractions for heterogeneous multicore/manycore computing. It is specified as an extension to the official grammar of Java 8, which can be obtained at `https://docs.oracle.com/javase/specs/jls/se8/html`.

---

1. Lexical Structure

```
Identifier -> IdentifierChars but not a Keyword or BooleanLiteral or NullLiteral
IdentifierChars -> JavaLetter {JavaLetterOrDigit}
JavaLetter -> any Unicode character that is a "Java letter"
JavaLetterOrDigit -> any Unicode character that is a "Java letter-or-digit"
Literal -> IntegerLiteral | FloatingPointLiteral | BooleanLiteral | CharacterLiteral | StringLiteral | NullLiteral
```

2. Types, Values, and Variables

```
Type -> PrimitiveType | ReferenceType
PrimitiveType -> {Annotation} NumericType | {Annotation} boolean
NumericType -> IntegralType | FloatingPointType
IntegralType -> byte | short | int | long | char
FloatingPointType -> float | double
ReferenceType -> ClassOrInterfaceType | TypeVariable | ArrayType
ClassOrInterfaceType -> ClassType | InterfaceType
ClassType -> {Annotation} Identifier [TypeArguments] |
            ClassOrInterfaceType . {Annotation} Identifier [TypeArguments]
InterfaceType -> ClassType
TypeVariable -> {Annotation} Identifier
ArrayType -> PrimitiveType Dims | ClassOrInterfaceType Dims | TypeVariable Dims
Dims -> {Annotation} [ ] {{Annotation} [ ]}
TypeParameter -> {TypeParameterModifier} Identifier [TypeBound]
TypeParameterModifier -> Annotation
TypeBound -> extends TypeVariable | extends ClassOrInterfaceType {AdditionalBound}
AdditionalBound -> & InterfaceType
TypeArguments -> < TypeArgumentList >
TypeArgumentList -> TypeArgument {, TypeArgument}
TypeArgument -> ReferenceType | Wildcard
Wildcard -> {Annotation} ? [WildcardBounds]
WildcardBounds -> extends ReferenceType | super ReferenceType
```

3. Names

```
TypeName -> Identifier | PackageOrTypeName . Identifier
PackageOrTypeName -> Identifier | PackageOrTypeName . Identifier
ExpressionName -> Identifier | AmbiguousName . Identifier
MethodName -> Identifier
PackageName -> Identifier | PackageName . Identifier
```

```
AmbiguousName -> Identifier | AmbiguousName . Identifier
```

4. Packages

```
CompilationUnit -> [PackageDeclaration] {ImportDeclaration} {TypeDeclaration}
PackageDeclaration -> {PackageModifier} package Identifier {. Identifier} ;
PackageModifier -> Annotation
ImportDeclaration -> SingleTypeImportDeclaration | TypeImportOnDemandDeclaration |
                     SingleStaticImportDeclaration | StaticImportOnDemandDeclaration
SingleTypeImportDeclaration -> import TypeName ;
TypeImportOnDemandDeclaration -> import PackageOrTypeName . * ;
SingleStaticImportDeclaration -> import static TypeName . Identifier ;
StaticImportOnDemandDeclaration -> import static TypeName . * ;
TypeDeclaration -> ClassDeclaration |  InterfaceDeclaration | ;
```

5. Classes

```
ClassDeclaration -> AcceleratorClassDeclaration | NormalClassDeclaration | EnumDeclaration
```

5.1. Normal Classes

```
ClassDeclaration -> AcceleratorClassDeclaration*** | NormalClassDeclaration | EnumDeclaration
NormalClassDeclaration -> {ClassModifier} class Identifier [TypeParameters] [Superclass] [Superinterfaces] ClassBody
ClassModifier -> Annotation | public | protected | private | abstract | static | final | strictfp
TypeParameters -> < TypeParameterList >
TypeParameterList -> TypeParameter {, TypeParameter}
Superclass -> extends ClassType
Superinterfaces -> implements InterfaceTypeList
InterfaceTypeList -> InterfaceType {, InterfaceType}
ClassBody -> { {ClassBodyDeclaration} }
ClassBodyDeclaration -> ClassMemberDeclaration | InstanceInitializer | StaticInitializer | ConstructorDeclaration
ClassMemberDeclaration -> FieldDeclaration | MethodDeclaration | ClassDeclaration | InterfaceDeclaration | ;
FieldDeclaration -> {FieldModifier} UnannType VariableDeclaratorList ;
FieldModifier -> Annotation | public | protected | private | static | final | transient | volatile
VariableDeclaratorList -> VariableDeclarator {, VariableDeclarator}
VariableDeclarator -> VariableDeclaratorId [= VariableInitializer]
VariableDeclaratorId -> Identifier [Dims]
VariableInitializer -> Expression | ArrayInitializer
UnannType -> UnannPrimitiveType | UnannReferenceType
UnannPrimitiveType -> NumericType | boolean
UnannReferenceType -> UnannClassOrInterfaceType | UnannTypeVariable | UnannArrayType
UnannClassOrInterfaceType -> UnannClassType | UnannInterfaceType
UnannClassType -> Identifier [TypeArguments] | UnannClassOrInterfaceType . {Annotation} Identifier [TypeArguments]
UnannInterfaceType -> UnannClassType
UnannTypeVariable -> Identifier
UnannArrayType -> UnannPrimitiveType Dims | UnannClassOrInterfaceType Dims | UnannTypeVariable Dims
MethodDeclaration -> {MethodModifier} MethodHeader MethodBody
MethodModifier -> Annotation | public | protected | private | abstract | static | final | synchronized | native | strictfp
MethodHeader -> Result MethodDeclarator [Throws] | TypeParameters {Annotation} Result MethodDeclarator [Throws]
Result -> UnannType | void
MethodDeclarator -> Identifier ( [FormalParameterList] ) [Dims]
FormalParameterList -> ReceiverParameter | FormalParameters , LastFormalParameter | LastFormalParameter
FormalParameters -> FormalParameter {, FormalParameter} | ReceiverParameter {, FormalParameter}
FormalParameter -> {VariableModifier} UnannType VariableDeclaratorId
VariableModifier -> Annotation | final
LastFormalParameter -> {VariableModifier} UnannType {Annotation} ... VariableDeclaratorId | FormalParameter
ReceiverParameter -> {Annotation} UnannType [Identifier .] this
Throws -> throws ExceptionTypeList
ExceptionTypeList -> ExceptionType {, ExceptionType}
ExceptionType -> ClassType | TypeVariable
MethodBody -> Block | ;
InstanceInitializer -> Block
StaticInitializer -> static Block
ConstructorDeclaration -> {ConstructorModifier} ConstructorDeclarator [Throws] ConstructorBody
ConstructorModifier -> Annotation | public | protected | private
```

```
ConstructorDeclarator -> [TypeParameters] SimpleTypeName ( [FormalParameterList] )
SimpleTypeName -> Identifier
ConstructorBody -> { [ExplicitConstructorInvocation] [BlockStatements] }
ExplicitConstructorInvocation -> [TypeArguments] this ( [ArgumentList] ) ; |
                                 [TypeArguments] super ( [ArgumentList] ) ;
                                 ExpressionName . [TypeArguments] super ( [ArgumentList] ) ;
                                 Primary . [TypeArguments] super ( [ArgumentList] ) ;
```

5.2. Accelerator Classes

Declaration of Accelerator Classes.

```
AcceleratorClassDeclaration*** -> {ClassModifier} accelerator class Identifier [TypeParameters]
                                                  [Superclass] [Superinterfaces] AcceleratorClassBody
AcceleratorClassBody*** -> { {AcceleratorClassBodyDeclaration***} }
AcceleratorClassBodyDeclaration*** -> InstanceInitializer | StaticInitializer | ConstructorDeclaration |
                                      AcceleratorClassMemberDeclaration*** |
AcceleratorClassMemberDeclaration*** -> FieldDeclaration | MethodDeclaration | ClassDeclaration | InterfaceDeclaration |
                                        KernelDeclaration*** | UnitDeclaration***;
```

Declaration of Units.

```
UnitDeclaration*** -> {UnitModifier***} unit Identifier UnitBody***
UnitModifier*** -> Annotation | abstract | final | parallel***
UnitBody*** -> { {UnitBodyDeclaration***} }
UnitBodyDeclaration*** -> InstanceInitializer | StaticInitializer | ConstructorDeclaration | UnitMemberDeclaration***
UnitMemberDeclaration*** -> FieldDeclaration | MethodDeclaration | KernelDeclaration*** ;
```

Declaratin of Kernel Methods.

```
KernelDeclaration*** -> {MethodModifier} KernelHeader*** KernelBody***
KernelHeader*** -> kernel KernelDeclarator***
KernelDeclarator*** -> Identifier ( [FormalParameterList] ) [Dims] {GridConfiguration***} {BlockConfiguration***}
KernelBody*** -> KernelBlock*** | ;
GridConfiguration*** -> grid  <<< KernelExpression*** , KernelExpression***, KernelExpression*** >>>
BlockConfiguration*** -> block <<< KernelExpression*** , KernelExpression***, KernelExpression*** >>>
```

5.3. Enumerations

```
EnumDeclaration -> {ClassModifier} enum Identifier [Superinterfaces] EnumBody
EnumBody -> { [EnumConstantList] [,] [EnumBodyDeclarations] }
EnumConstantList -> EnumConstant {, EnumConstant}
EnumConstant -> {EnumConstantModifier} Identifier [( [ArgumentList] )] [ClassBody]
EnumConstantModifier -> Annotation
EnumBodyDeclarations -> ; {ClassBodyDeclaration}
```

6. Interfaces

```
InterfaceDeclaration -> AcceleratorInterfaceDeclaration*** | NormalInterfaceDeclaration | AnnotationTypeDeclaration
```

6.1. Normal Interfaces

```
NormalInterfaceDeclaration ->  {InterfaceModifier} interface Identifier [TypeParameters] [ExtendsInterfaces] InterfaceBody
InterfaceModifier -> Annotation | public | protected | private | abstract | static | strictfp
ExtendsInterfaces -> extends InterfaceTypeList
InterfaceBody -> { {InterfaceMemberDeclaration} }
InterfaceMemberDeclaration -> ConstantDeclaration | InterfaceMethodDeclaration | ClassDeclaration | InterfaceDeclaration ;
ConstantDeclaration -> {ConstantModifier} UnannType VariableDeclaratorList ;
ConstantModifier -> Annotation | public | static | final
InterfaceMethodDeclaration -> {InterfaceMethodModifier} MethodHeader MethodBody
InterfaceMethodModifier -> Annotation | public | abstract | default | static | strictfp | parallel***
```

6.2. Accelerator Interfaces

```
AcceleratorInterfaceDeclaration*** ->  {InterfaceModifier} accelerator interface Identifier
                                                            [TypeParameters] [ExtendsInterfaces] AcceleratorInterfaceBody
AcceleratorInterfaceBody*** -> { {AcceleratorInterfaceMemberDeclaration***} }
AcceleratorInterfaceMemberDeclaration*** ->
                   ConstantDeclaration | InterfaceMethodDeclaration | ClassDeclaration | InterfaceDeclaration |
                   InterfaceKernelDeclaration*** | AcceleratorClassDeclaration*** | AcceleratorInterfaceDeclaration*** |
                   UnitInterfaceDeclaration*** ;
UnitInterfaceDeclaration*** -> {InterfaceModifier} unit Identifier UnitInterfaceBody***
UnitInterfaceBody*** -> { {UnitInterfaceMemberDeclaration***} }
UnitInterfaceMemberDeclaration*** -> ConstantDeclaration | InterfaceMethodDeclaration | ClassDeclaration |
                                     InterfaceKernelDeclaration*** | InterfaceDeclaration ;
InterfaceKernelDeclaration*** -> {InterfaceMethodModifier} KernelHeader*** KernelBody***
```

6.3. Type Annotations

```
AnnotationTypeDeclaration -> {InterfaceModifier} @ interface Identifier AnnotationTypeBody
AnnotationTypeBody -> { {AnnotationTypeMemberDeclaration} }
AnnotationTypeMemberDeclaration -> AnnotationTypeElementDeclaration | ConstantDeclaration |
                                  ClassDeclaration | InterfaceDeclaration ;
AnnotationTypeElementDeclaration -> {AnnotationTypeElementModifier} UnannType Identifier ( ) [Dims] [DefaultValue] ;
AnnotationTypeElementModifier -> Annotation | public | abstract
DefaultValue -> default ElementValue
Annotation -> NormalAnnotation | MarkerAnnotation | SingleElementAnnotation
NormalAnnotation -> @ TypeName ( [ElementValuePairList] )
ElementValuePairList -> ElementValuePair {, ElementValuePair}
ElementValuePair -> Identifier = ElementValue
ElementValue -> ConditionalExpression | ElementValueArrayInitializer | Annotation
ElementValueArrayInitializer -> { [ElementValueList] [,] }
ElementValueList -> ElementValue {, ElementValue}
MarkerAnnotation -> @ TypeName
SingleElementAnnotation -> @ TypeName ( ElementValue )
```

7. Arrays

```
ArrayInitializer -> { [VariableInitializerList] [,] }
VariableInitializerList -> VariableInitializer {, VariableInitializer}
```

8. Blocks and Statements

```
Block -> { [BlockStatements] }
BlockStatements -> BlockStatement {BlockStatement}
BlockStatement -> LocalVariableDeclarationStatement | ClassDeclaration | Statement
LocalVariableDeclarationStatement -> LocalVariableDeclaration ;
LocalVariableDeclaration -> {VariableModifier} UnannType VariableDeclaratorList
Statement -> StatementWithoutTrailingSubstatement | LabeledStatement |
             IfThenStatement | IfThenElseStatement | WhileStatement | ForStatement
StatementNoShortIf -> StatementWithoutTrailingSubstatement | LabeledStatementNoShortIf |
                      IfThenElseStatementNoShortIf | WhileStatementNoShortIf | ForStatementNoShortIf
StatementWithoutTrailingSubstatement -> Block | EmptyStatement | ExpressionStatement | AssertStatement |
                                        SwitchStatement | DoStatement | BreakStatement | ContinueStatement |
                                        ReturnStatement | SynchronizedStatement | ThrowStatement | TryStatement
EmptyStatement -> ;
LabeledStatement ->
Identifier -> Statement
LabeledStatementNoShortIf ->
Identifier -> StatementNoShortIf
ExpressionStatement -> StatementExpression ;
StatementExpression -> Assignment | AsyncStatement*** | PreIncrementExpression | PreDecrementExpression | PostIncrementExpression |
                       PostDecrementExpression | MethodInvocation | ClassInstanceCreationExpression
AsyncStatement*** -> async Assignment
IfThenStatement -> if ( Expression ) Statement
IfThenElseStatement -> if ( Expression ) StatementNoShortIf else Statement
IfThenElseStatementNoShortIf -> if ( Expression ) StatementNoShortIf else StatementNoShortIf
AssertStatement -> assert Expression ;
assert Expression -> Expression ;
```

```
SwitchStatement -> switch ( Expression ) SwitchBlock
SwitchBlock -> { {SwitchBlockStatementGroup} {SwitchLabel} }
SwitchBlockStatementGroup -> SwitchLabels BlockStatements
SwitchLabels -> SwitchLabel {SwitchLabel}
SwitchLabel ->
case ConstantExpression ->
case EnumConstantName ->
default ->
EnumConstantName -> Identifier
WhileStatement -> while ( Expression ) Statement
WhileStatementNoShortIf -> while ( Expression ) StatementNoShortIf
DoStatement -> do Statement while ( Expression ) ;
ForStatement -> BasicForStatement | EnhancedForStatement
ForStatementNoShortIf -> BasicForStatementNoShortIf | EnhancedForStatementNoShortIf
BasicForStatement -> for ( [ForInit] ; [Expression] ; [ForUpdate] ) Statement
BasicForStatementNoShortIf -> for ( [ForInit] ; [Expression] ; [ForUpdate] ) StatementNoShortIf
ForInit -> StatementExpressionList | LocalVariableDeclaration
ForUpdate -> StatementExpressionList
StatementExpressionList -> StatementExpression {, StatementExpression}
EnhancedForStatement -> for ( {VariableModifier} UnannType VariableDeclaratorId : Expression ) Statement
EnhancedForStatementNoShortIf -> for ( {VariableModifier} UnannType VariableDeclaratorId : Expression ) StatementNoShortIf
BreakStatement -> break [Identifier] ;
ContinueStatement -> continue [Identifier] ;
ReturnStatement -> return [Expression] ;
ThrowStatement -> throw Expression ;
SynchronizedStatement -> synchronized ( Expression ) Block
TryStatement -> try Block Catches | try Block [Catches] Finally | TryWithResourcesStatement
Catches -> CatchClause {CatchClause}
CatchClause -> catch ( CatchFormalParameter ) Block
CatchFormalParameter -> {VariableModifier} CatchType VariableDeclaratorId
CatchType -> UnannClassType {| ClassType}
Finally -> finally Block
TryWithResourcesStatement -> try ResourceSpecification Block [Catches] [Finally]
ResourceSpecification -> ( ResourceList [;] )
ResourceList -> Resource {; Resource}
Resource -> {VariableModifier} UnannType VariableDeclaratorId = Expression
```

9. Expressions

```
Primary -> PrimaryNoNewArray | ArrayCreationExpression
PrimaryNoNewArray -> Literal | ClassLiteral | this | TypeName . this | ( Expression ) |
                     ClassInstanceCreationExpression | FieldAccess | ArrayAccess |
                     MethodInvocation | MethodReference
ClassLiteral -> TypeName {[ ]} . class | NumericType {[ ]} . class | boolean {[ ]} . class | void . class
ClassInstanceCreationExpression -> UnqualifiedClassInstanceCreationExpression |
                                   ExpressionName . UnqualifiedClassInstanceCreationExpression |
                                   Primary . UnqualifiedClassInstanceCreationExpression
UnqualifiedClassInstanceCreationExpression ->
                   new [TypeArguments] ClassOrInterfaceTypeToInstantiate ( [ArgumentList] ) [@ Expression]*** [ClassBody]
ClassOrInterfaceTypeToInstantiate -> {Annotation} Identifier {. {Annotation} Identifier} [TypeArgumentsOrDiamond]
TypeArgumentsOrDiamond -> TypeArguments | <>
FieldAccess -> Primary . Identifier | super . Identifier | TypeName . super . Identifier
ArrayAccess -> ExpressionName [ Expression ] | PrimaryNoNewArray [ Expression ]
MethodInvocation -> MethodName ( [ArgumentList] )  |
                    TypeName . [TypeArguments] Identifier ( [ArgumentList] ) |
                    ExpressionName . [TypeArguments] Identifier ( [ArgumentList] ) |
                    Primary . [TypeArguments] Identifier ( [ArgumentList] ) |
                    super . [TypeArguments] Identifier ( [ArgumentList] ) |
                    TypeName . super . [TypeArguments] Identifier ( [ArgumentList] )
ArgumentList -> Expression {, Expression}
MethodReference -> ExpressionName :: [TypeArguments] Identifier |
                   ReferenceType :: [TypeArguments] Identifier |
                   Primary :: [TypeArguments] Identifier |
                   super :: [TypeArguments] Identifier |
                   TypeName . super :: [TypeArguments] Identifier |
                   ClassType :: [TypeArguments] new |
                   ArrayType :: new
ArrayCreationExpression -> new PrimitiveType DimExprs [Dims] |
                           new ClassOrInterfaceType DimExprs [Dims] |
                           new PrimitiveType Dims ArrayInitializer |
                           new ClassOrInterfaceType Dims ArrayInitializer
DimExprs -> DimExpr {DimExpr}
DimExpr -> {Annotation} [ Expression ]
```

```
Expression -> LambdaExpression | AssignmentExpression
LambdaExpression ->
LambdaParameters -> LambdaBody
LambdaParameters -> Identifier | ( [FormalParameterList] ) | ( InferredFormalParameterList )
InferredFormalParameterList -> Identifier {, Identifier}
LambdaBody -> Expression | Block
AssignmentExpression -> ConditionalExpression | Assignment
Assignment -> LeftHandSide AssignmentOperator Expression
LeftHandSide -> ExpressionName | FieldAccess | ArrayAccess
AssignmentOperator -> = | *= | /= | %= | += | -= | <<= | >>= | >>>= | &= | ^= | |=
ConditionalExpression -> ConditionalOrExpression
ConditionalOrExpression ? Expression -> ConditionalExpression
ConditionalOrExpression ? Expression -> LambdaExpression
ConditionalOrExpression -> ConditionalAndExpression | ConditionalOrExpression || ConditionalAndExpression
ConditionalAndExpression -> InclusiveOrExpression | ConditionalAndExpression && InclusiveOrExpression
InclusiveOrExpression -> ExclusiveOrExpression | InclusiveOrExpression | ExclusiveOrExpression
ExclusiveOrExpression -> AndExpression | ExclusiveOrExpression ^ AndExpression
AndExpression -> EqualityExpression | AndExpression & EqualityExpression
EqualityExpression -> RelationalExpression |
                      EqualityExpression == RelationalExpression |
                      EqualityExpression != RelationalExpression
RelationalExpression -> ShiftExpression |
                        RelationalExpression < ShiftExpression |
                        RelationalExpression > ShiftExpression |
                        RelationalExpression <= ShiftExpression |
                        RelationalExpression >= ShiftExpression |
                        RelationalExpression instanceof ReferenceType
ShiftExpression -> AdditiveExpression |
                   ShiftExpression << AdditiveExpression |
                   ShiftExpression >> AdditiveExpression |
                   ShiftExpression >>> AdditiveExpression
AdditiveExpression -> MultiplicativeExpression |
                      AdditiveExpression + MultiplicativeExpression |
                      AdditiveExpression - MultiplicativeExpression
MultiplicativeExpression -> UnaryExpression |
                            MultiplicativeExpression * UnaryExpression |
                            MultiplicativeExpression / UnaryExpression |
                            MultiplicativeExpression % UnaryExpression
UnaryExpression -> PreIncrementExpression | PreDecrementExpression |
                   + UnaryExpression | - UnaryExpression | UnaryExpressionNotPlusMinus
PreIncrementExpression -> ++ UnaryExpression
PreDecrementExpression -> -- UnaryExpression
UnaryExpressionNotPlusMinus -> PostfixExpression | ~ UnaryExpression | ! UnaryExpression | CastExpression
PostfixExpression -> Primary | ExpressionName | PostIncrementExpression | PostDecrementExpression
PostIncrementExpression -> PostfixExpression ++
PostDecrementExpression -> PostfixExpression --
CastExpression -> ( PrimitiveType ) UnaryExpression |
                  ( ReferenceType {AdditionalBound} ) UnaryExpressionNotPlusMinus |
                  ( ReferenceType {AdditionalBound} ) LambdaExpression
ConstantExpression -> Expression
```

10. Kernel Blocks and Statements

```
KernelBlock*** -> { [BlockStatements***] }
BlockStatements*** -> BlockStatement*** {BlockStatement***}
BlockStatement*** -> LocalVariableDeclarationStatement*** | Statement***
LocalVariableDeclarationStatement*** -> LocalVariableDeclaration*** ;
LocalVariableDeclaration*** -> NumericType VariableDeclaratorList

Statement*** -> StatementWithoutTrailingSubstatement*** | LabeledStatement |
            IfThenStatement*** | IfThenElseStatement*** | WhileStatement*** | ForStatement***
StatementNoShortIf*** -> StatementWithoutTrailingSubstatement*** | LabeledStatementNoShortIf |
                      IfThenElseStatementNoShortIf | WhileStatementNoShortIf | ForStatementNoShortIf
StatementWithoutTrailingSubstatement*** -> KernelBlock*** | EmptyStatement | ExpressionStatement*** |
                                           SwitchStatement*** | DoStatement*** | BreakStatement | ContinueStatement |
                                           ReturnStatement***
LabeledStatementNoShortIf ->
ExpressionStatement*** -> StatementExpression*** ;
StatementExpression*** -> Assignment | PreIncrementExpression | PreDecrementExpression | PostIncrementExpression |
                       PostDecrementExpression | KernelMethodInvocation*** |
IfThenStatement*** -> if ( KernelExpression*** ) Statement***
IfThenElseStatement*** -> if ( KernelExpression*** ) StatementNoShortIf*** else Statement***
```

```
IfThenElseStatementNoShortIf*** -> if ( KernelExpression*** ) StatementNoShortIf*** else StatementNoShortIf***
SwitchStatement*** -> switch ( KernelExpression*** ) SwitchBlock***
SwitchBlock*** -> { {SwitchBlockStatementGroup***} {SwitchLabel} }
SwitchBlockStatementGroup*** -> SwitchLabels BlockStatements***
SwitchLabels -> SwitchLabel {SwitchLabel}
WhileStatement*** -> while*** ( Expression*** ) Statement***
WhileStatementNoShortIf*** -> while ( KernelExpression*** ) StatementNoShortIf***
DoStatement*** -> do Statement*** while ( KernelExpression*** ) ;
ForStatement*** -> for ( [ForInit***] ; [KernelExpression***] ; [ForUpdate***] ) Statement***
ForStatementNoShortIf*** -> for ( [ForInit***] ; [KernelExpression***] ; [ForUpdate***] ) StatementNoShortIf***
ForInit*** -> StatementExpressionList*** | LocalVariableDeclaration***
ForUpdate*** -> StatementExpressionList***
StatementExpressionList*** -> StatementExpression*** {, StatementExpression***}
ReturnStatement*** -> return [KernelExpression***] ;
```

11. Kernel Expressions

```
Primary*** -> PrimaryNoNewArray*** | ArrayCreationExpression***
PrimaryNoNewArray*** -> Literal | ( KernelExpression*** ) | FieldAccess*** | ArrayAccess*** | KernelMethodInvocation***
FieldAccess*** -> Primary . Identifier
ArrayAccess*** -> ExpressionName [ Expression ] | PrimaryNoNewArray*** [ KernelExpression*** ]
KernelMethodInvocation*** -> MethodName ( [ArgumentList***] )
ArgumentList*** -> KernelExpression*** {, KernelExpression***}
ArrayCreationExpression*** -> new PrimitiveType DimExprs [Dims]
                           new PrimitiveType Dims ArrayInitializer
DimExprs -> DimExpr {DimExpr}
DimExpr -> {Annotation} [ Expression ]

Expression*** -> AssignmentExpression***
InferredFormalParameterList -> Identifier {, Identifier}
AssignmentExpression*** -> ConditionalExpression*** | Assignment***
Assignment*** -> LeftHandSide AssignmentOperator Expression***
LeftHandSide -> ExpressionName | FieldAccess | ArrayAccess
ConditionalExpression*** -> ConditionalOrExpression***
ConditionalOrExpression*** ? Expression*** -> ConditionalExpression***
ConditionalOrExpression*** ? Expression*** -> LambdaExpression
ConditionalOrExpression*** -> ConditionalAndExpression | ConditionalOrExpression*** || ConditionalAndExpression***
ConditionalAndExpression*** -> InclusiveOrExpression*** | ConditionalAndExpression*** && InclusiveOrExpression***
InclusiveOrExpression*** -> ExclusiveOrExpression | InclusiveOrExpression | ExclusiveOrExpression***
ExclusiveOrExpression*** -> AndExpression*** | ExclusiveOrExpression*** ^ AndExpression***
AndExpression -> EqualityExpression | AndExpression & EqualityExpression
EqualityExpression -> RelationalExpression |
                      EqualityExpression == RelationalExpression |
                      EqualityExpression != RelationalExpression
RelationalExpression -> ShiftExpression |
                        RelationalExpression < ShiftExpression |
                        RelationalExpression > ShiftExpression |
                        RelationalExpression <= ShiftExpression |
                        RelationalExpression >= ShiftExpression |
                        RelationalExpression instanceof ReferenceType
ShiftExpression -> AdditiveExpression |
                   ShiftExpression << AdditiveExpression |
                   ShiftExpression >> AdditiveExpression |
                   ShiftExpression >>> AdditiveExpression
AdditiveExpression -> MultiplicativeExpression |
                      AdditiveExpression + MultiplicativeExpression |
                      AdditiveExpression - MultiplicativeExpression
MultiplicativeExpression -> UnaryExpression |
                            MultiplicativeExpression * UnaryExpression |
                            MultiplicativeExpression / UnaryExpression |
                            MultiplicativeExpression % UnaryExpression
UnaryExpression -> PreIncrementExpression | PreDecrementExpression |
                   + UnaryExpression | - UnaryExpression | UnaryExpressionNotPlusMinus
PreIncrementExpression -> ++ UnaryExpression
PreDecrementExpression -> -- UnaryExpression
UnaryExpressionNotPlusMinus -> PostfixExpression | ~ UnaryExpression | ! UnaryExpression | CastExpression
PostfixExpression -> Primary | ExpressionName | PostIncrementExpression | PostDecrementExpression
PostIncrementExpression -> PostfixExpression ++
PostDecrementExpression -> PostfixExpression --
CastExpression -> ( PrimitiveType ) UnaryExpression |
                  ( ReferenceType {AdditionalBound} ) UnaryExpressionNotPlusMinus |
                  ( ReferenceType {AdditionalBound} ) LambdaExpression
ConstantExpression -> Expression
```