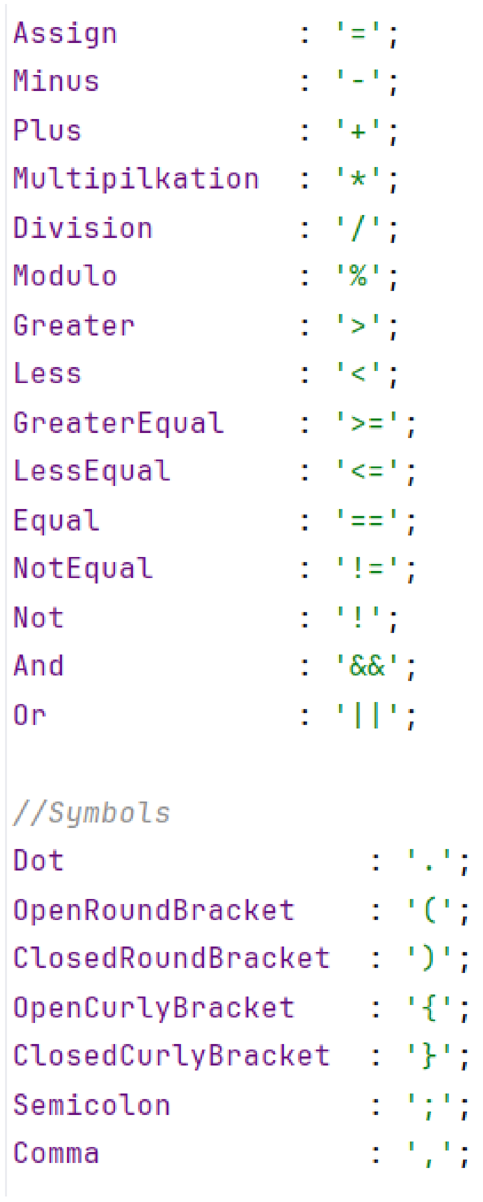
DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart
Campus Horb

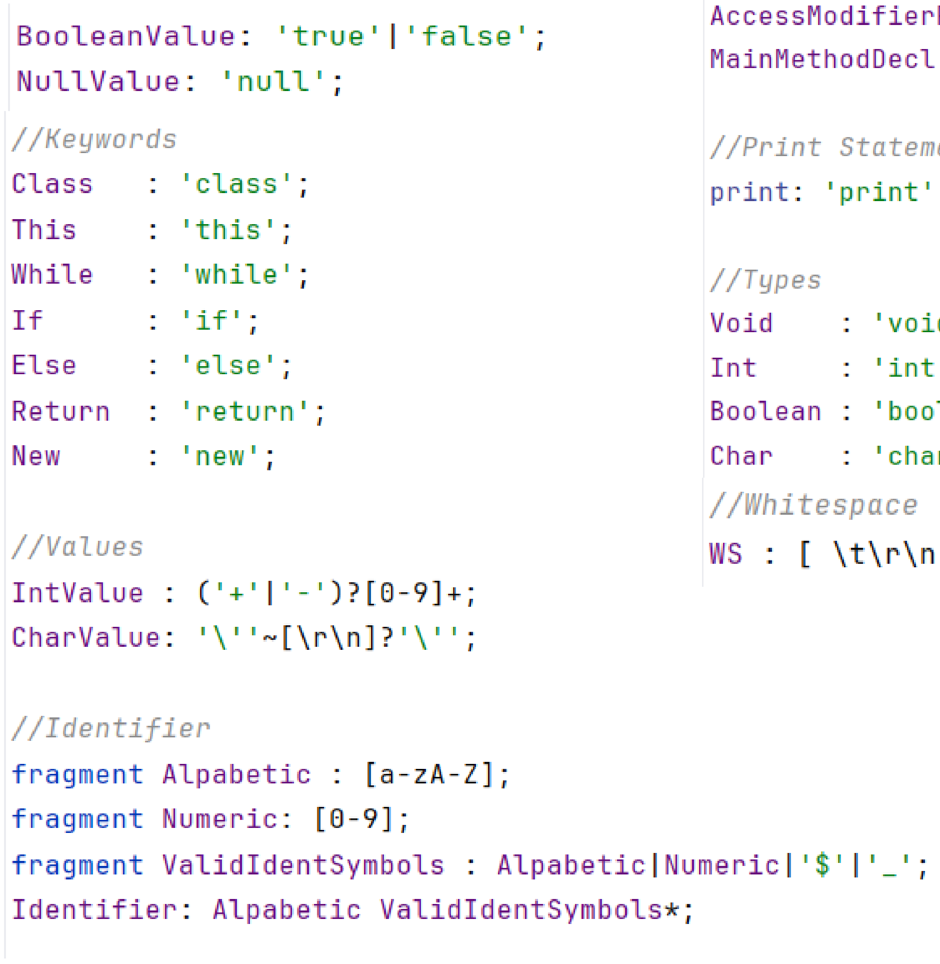
NichtHaskell

Compilerbau-Projekt

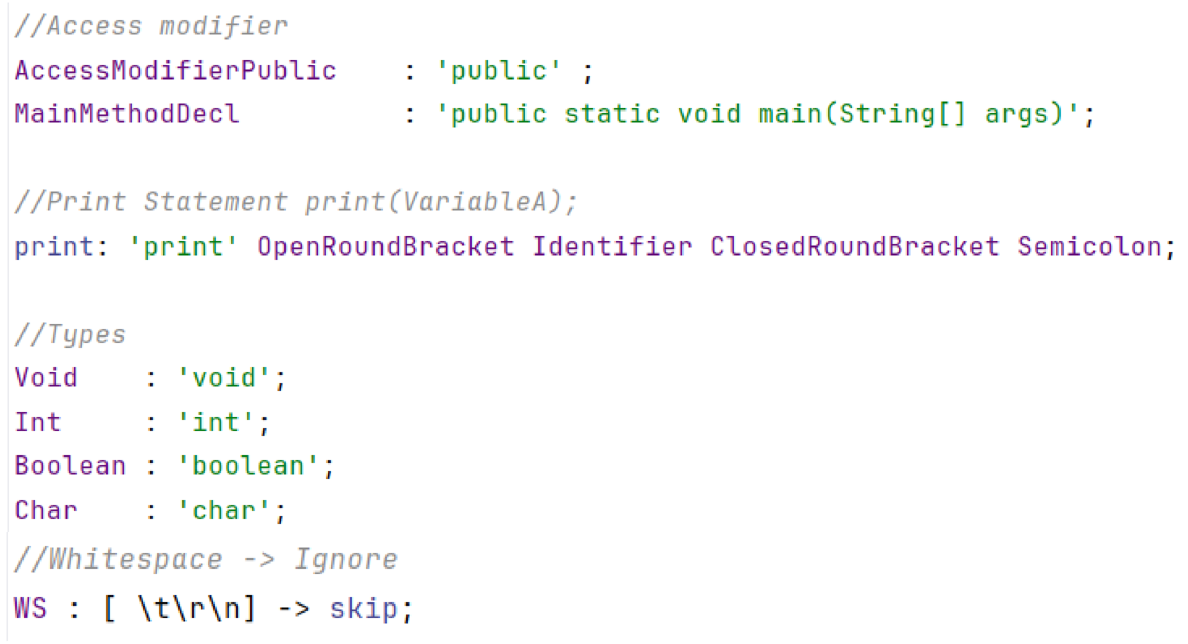
Scanner



```
Assign          : '=';
Minus           : '-';
Plus            : '+';
Multipilkation  : '*';
Division        : '/';
Modulo          : '%';
Greater         : '>';
Less            : '<';
GreaterEqual    : '>=';
LessEqual       : '<=';
Equal           : '==';
NotEqual        : '!=';
Not             : '!';
And             : '&&';
Or              : '||';

//Symbols
Dot                 : '.';
OpenRoundBracket    : '(';
ClosedRoundBracket  : ')';
OpenCurlyBracket    : '{';
ClosedCurlyBracket  : '}';
Semicolon           : ';';
Comma               : ',';
```

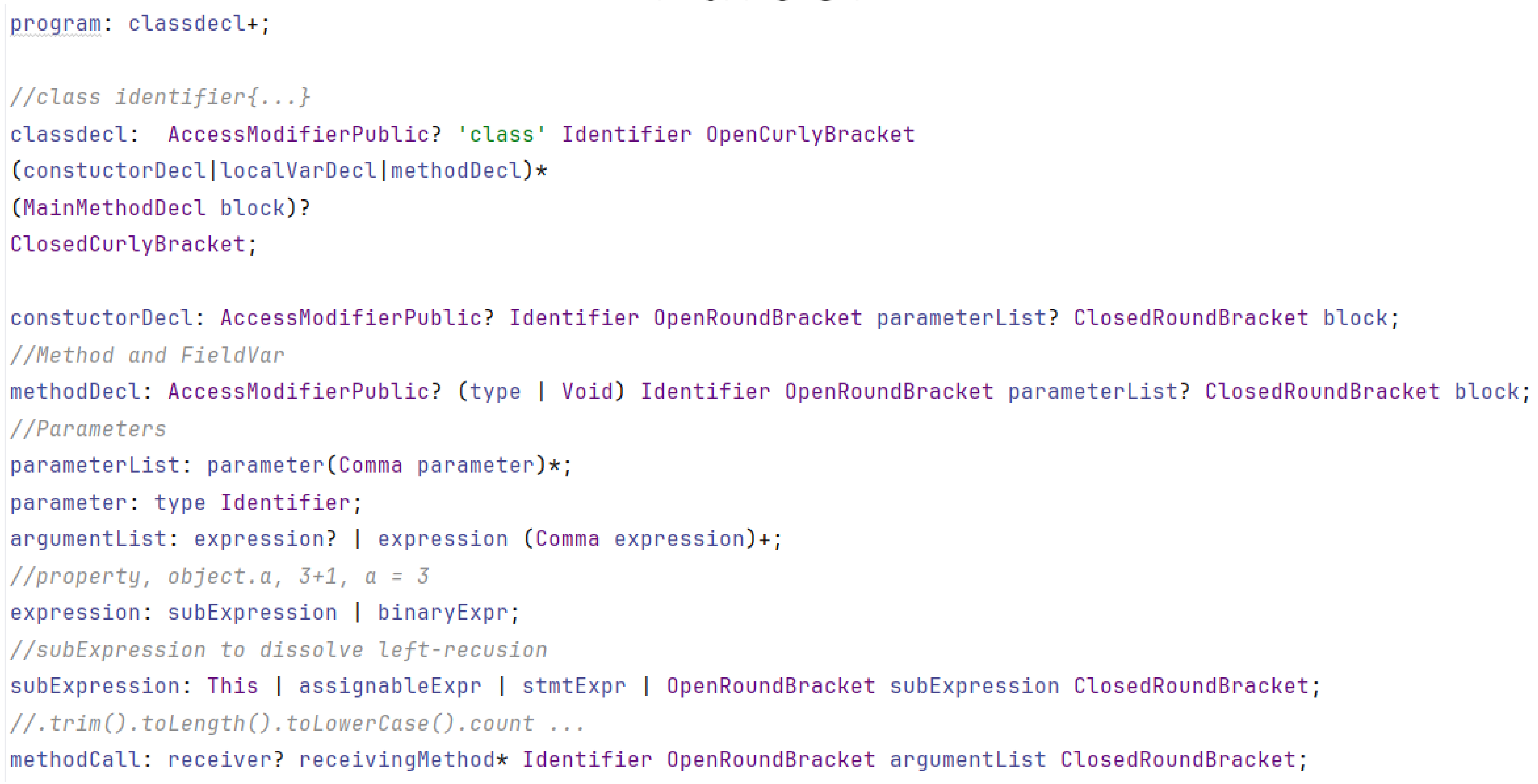


```
BooleanValue: 'true'|'false';
NullValue: 'null';

//Keywords
Class   : 'class';
This    : 'this';
While   : 'while';
If      : 'if';
Else    : 'else';
Return  : 'return';
New     : 'new';

//Values
IntValue : ('+'|'-')?[0-9]+;
CharValue: '\''~[\r\n]?'\'';

//Identifier
fragment Alpabetic : [a-zA-Z];
fragment Numeric: [0-9];
fragment ValidIdentSymbols : Alpabetic|Numeric|'$'|'_';
Identifier: Alpabetic ValidIdentSymbols*;
```



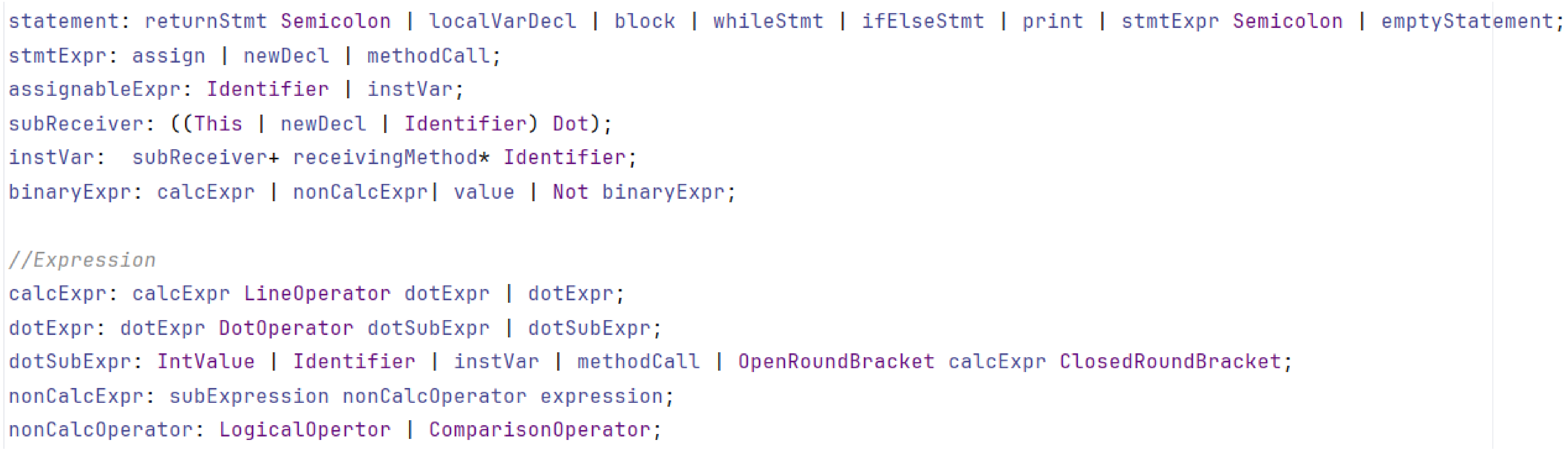
```
//Access modifier
AccessModifierPublic    : 'public' ;
MainMethodDecl          : 'public static void main(String[] args)';

//Print Statement print(VariableA);
print: 'print' OpenRoundBracket Identifier ClosedRoundBracket Semicolon;

//Types
Void    : 'void';
Int     : 'int';
Boolean : 'boolean';
Char    : 'char';
//Whitespace -> Ignore
WS : [ \t\r\n] -> skip;
```

Parser



```
program: classdecl+;

//class identifier{...}
classdecl:  AccessModifierPublic? 'class' Identifier OpenCurlyBracket
(constuctorDecl|localVarDecl|methodDecl)*
(MainMethodDecl block)?
ClosedCurlyBracket;

constuctorDecl: AccessModifierPublic? Identifier OpenRoundBracket parameterList? ClosedRoundBracket block;
//Method and FieldVar
methodDecl: AccessModifierPublic? (type | Void) Identifier OpenRoundBracket parameterList? ClosedRoundBracket block;
//Parameters
parameterList: parameter(Comma parameter)*;
parameter: type Identifier;
argumentList: expression? | expression (Comma expression)+;
//property, object.a, 3+1, a = 3
expression: subExpression | binaryExpr;
//subExpression to dissolve left-recusion
subExpression: This | assignableExpr | stmtExpr | OpenRoundBracket subExpression ClosedRoundBracket;
//.trim().toLength().toLowerCase().count ...
methodCall: receiver? receivingMethod* Identifier OpenRoundBracket argumentList ClosedRoundBracket;
```

Parser

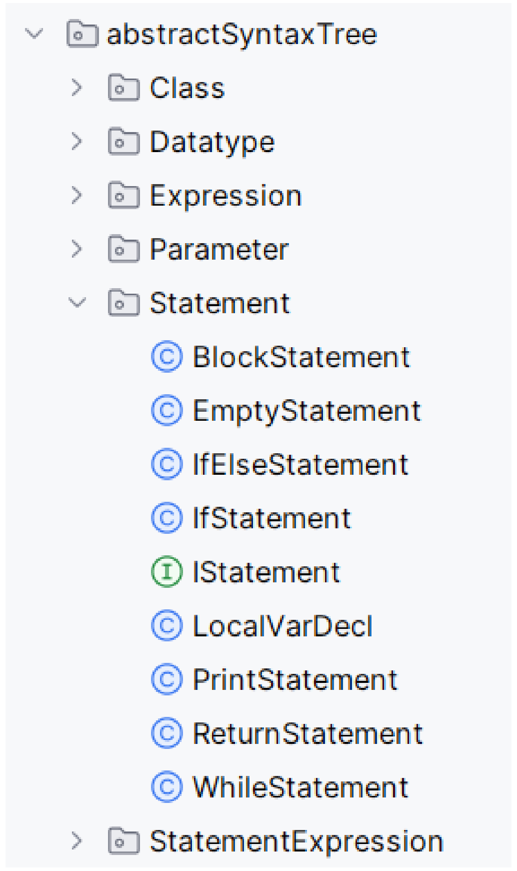


```
statement: returnStmt Semicolon | localVarDecl | block | whileStmt | ifElseStmt | print | stmtExpr Semicolon | emptyStatement;
stmtExpr: assign | newDecl | methodCall;
assignableExpr: Identifier | instVar;
subReceiver: ((This | newDecl | Identifier) Dot);
instVar:  subReceiver+ receivingMethod* Identifier;
binaryExpr: calcExpr | nonCalcExpr| value | Not binaryExpr;

//Expression
calcExpr: calcExpr LineOperator dotExpr | dotExpr;
dotExpr: dotExpr DotOperator dotSubExpr | dotSubExpr;
dotSubExpr: IntValue | Identifier | instVar | methodCall | OpenRoundBracket calcExpr ClosedRoundBracket;
nonCalcExpr: subExpression nonCalcOperator expression;
nonCalcOperator: LogicalOpertor | ComparisonOperator;
```

Parser

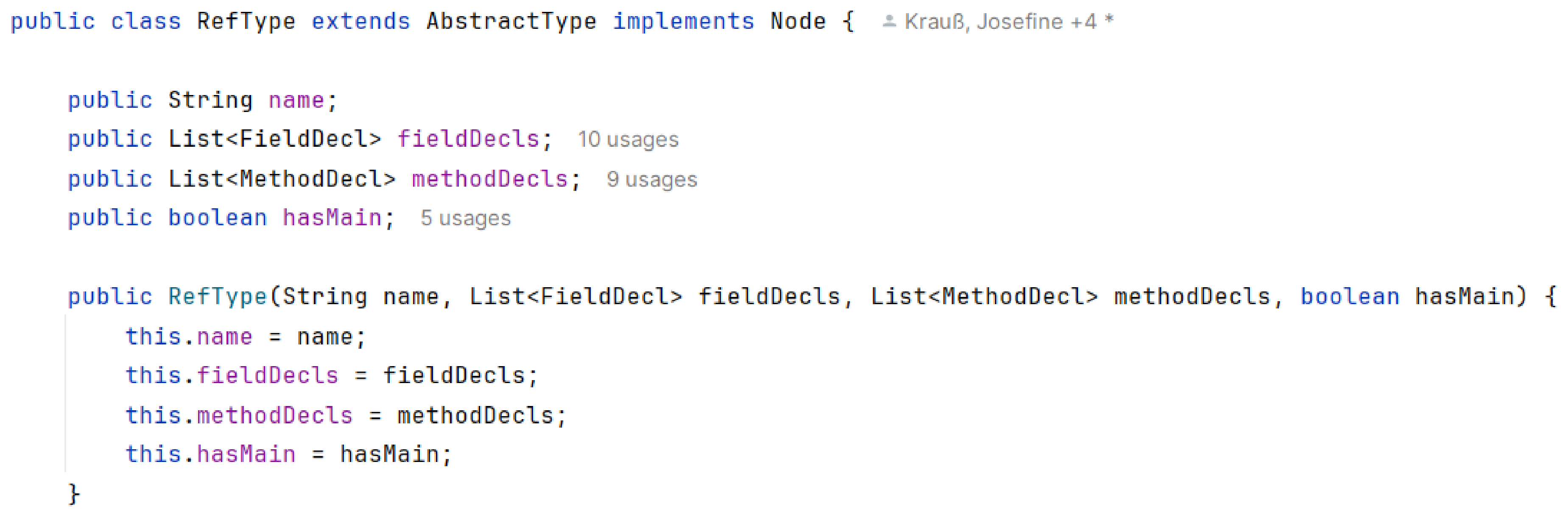
```
//Statements
returnStmt: Return (expression)?;
localVarDecl: AccessModifierPublic? type Identifier (Assign expression)? Semicolon;
block: OpenCurlyBracket statement* ClosedCurlyBracket;
whileStmt: While OpenRoundBracket expression ClosedRoundBracket statement;
ifElseStmt: ifStmt elseStmt?;
ifStmt: If OpenRoundBracket expression ClosedRoundBracket statement;
elseStmt: Else statement;
assign: assignableExpr Assign expression;
newDecl: New Identifier OpenRoundBracket argumentList ClosedRoundBracket;
receiver: ((This | instVar | newDecl | Identifier) Dot);
receivingMethod: Identifier OpenRoundBracket argumentList ClosedRoundBracket Dot;
emptyStatement : Semicolon;

type: Int | Boolean  | Char | Identifier;
value: IntValue | BooleanValue | CharValue | NullValue;
```

AST



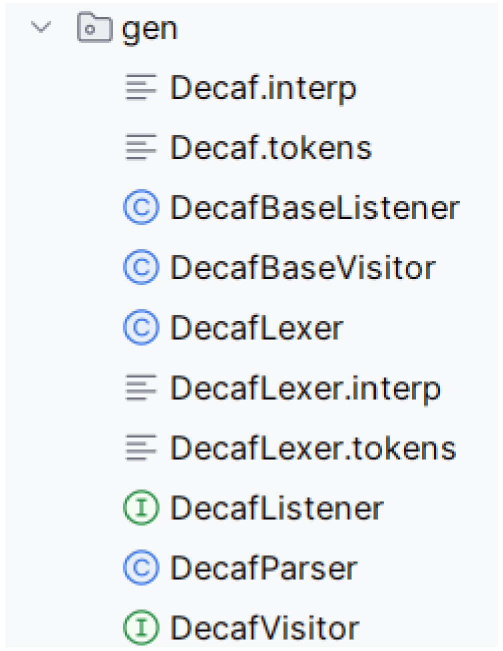
```
abstractSyntaxTree
    Class
    Datatype
    Expression
    Parameter
    Statement
        BlockStatement
        EmptyStatement
        IfElseStatement
        IfStatement
        IStatement
        LocalVarDecl
        PrintStatement
        ReturnStatement
        WhileStatement
    StatementExpression
```

AST



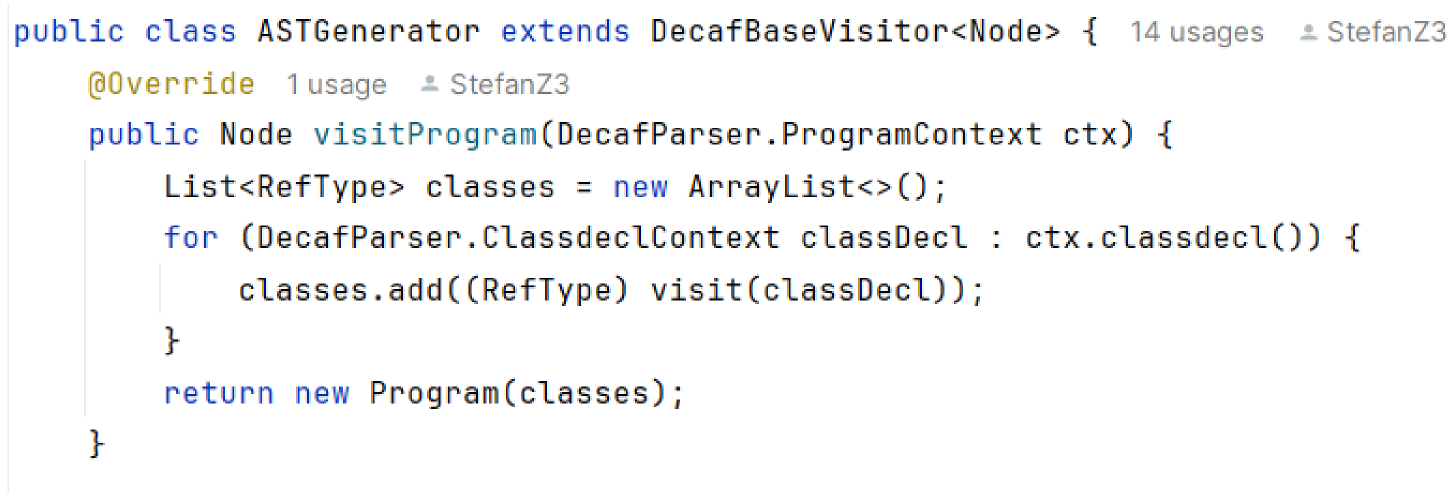
```
public class RefType extends AbstractType implements Node {  Krauß, Josefine +4 *

    public String name;
    public List<FieldDecl> fieldDecls;  10 usages
    public List<MethodDecl> methodDecls;  9 usages
    public boolean hasMain;  5 usages

    public RefType(String name, List<FieldDecl> fieldDecls, List<MethodDecl> methodDecls, boolean hasMain) {
        this.name = name;
        this.fieldDecls = fieldDecls;
        this.methodDecls = methodDecls;
        this.hasMain = hasMain;
    }
```

AST



gen
- Decaf.interp
- Decaf.tokens
- DecafBaseListener
- DecafBaseVisitor
- DecafLexer
- DecafLexer.interp
- DecafLexer.tokens
- DecafListener
- DecafParser
- DecafVisitor

AST



```
public class ASTGenerator extends DecafBaseVisitor<Node> {  14 usages  StefanZ3
    @Override  1 usage  StefanZ3
    public Node visitProgram(DecafParser.ProgramContext ctx) {
        List<RefType> classes = new ArrayList<>();
        for (DecafParser.ClassdeclContext classDecl : ctx.classdecl()) {
            classes.add((RefType) visit(classDecl));
        }
        return new Program(classes);
    }
```

TypeCheck

- Startpunkt im AST: Program
  - typeCheck-Methoden jeweils auf untergeordnetem Element aufrufen

- IStatement, IExpression, IStatementExpression, IDatatype:
  `typeCheck(/*...*/ methodContext, /*...*/ typeContext, /*...*/ localVars)`

- TypeCheckResult enthält ermittelten Typ
  - Rückgabewert der typeCheck-Methoden
  - In abstrakter Klasse AbstractType gespeichert
    - Superklasse aller Klassen mit TypeCheck
    - Zugriff durch Codegenerierung durch getTypeCheckResult()

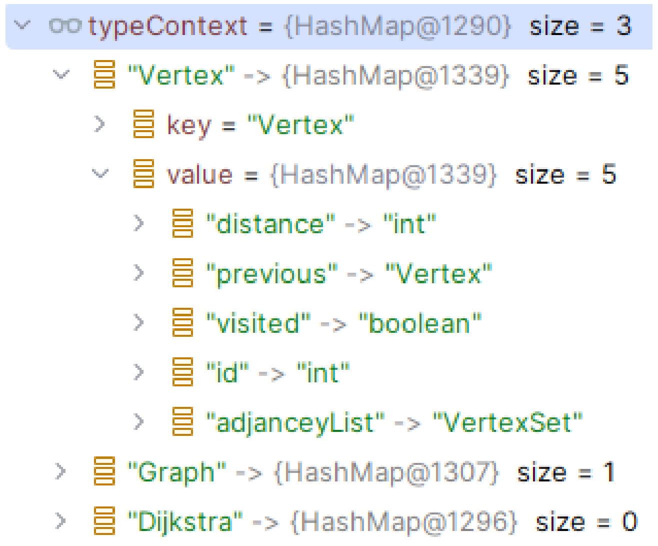
- TypeCheckException: Fehler der Inputdatei

- TypeCheckHelper: String upperBound, boolean typeExists

TypeCheck

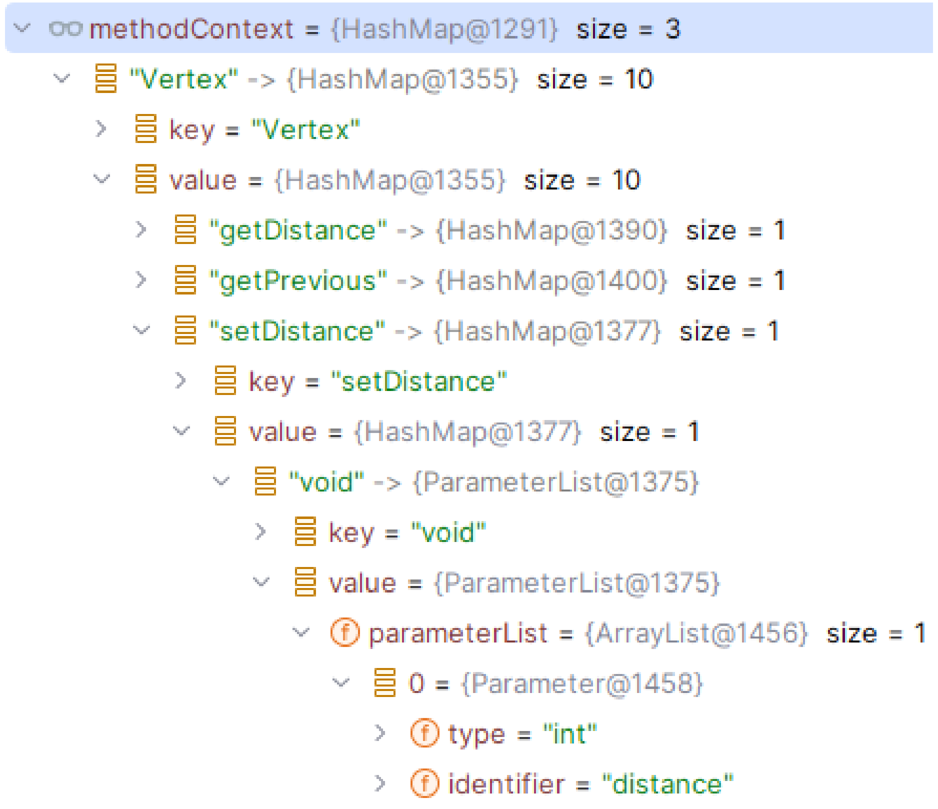
```
Program.typeCheck {
            Typ-Kontext
            Methoden-Kontext
            Main vorhanden
            Each RefType: typeCheck
}
```

Typ-Kontext:
Klassenname → Feldname → Typbezeichnung



```
∨ typeContext = {HashMap@1290}  size = 3
    ∨ "Vertex" -> {HashMap@1339}  size = 5
        > key = "Vertex"
        ∨ value = {HashMap@1339}  size = 5
            > "distance" -> "int"
            > "previous" -> "Vertex"
            > "visited" -> "boolean"
            > "id" -> "int"
            > "adjanceyList" -> "VertexSet"
    > "Graph" -> {HashMap@1307}  size = 1
    > "Dijkstra" -> {HashMap@1296}  size = 0
```

Methoden-Kontext:
Klassenname
→ Methodenname
→ Typbezeichner
→ List<Parameter>



```
∨ methodContext = {HashMap@1291}  size = 3
    ∨ "Vertex" -> {HashMap@1355}  size = 10
        > key = "Vertex"
        ∨ value = {HashMap@1355}  size = 10
            > "getDistance" -> {HashMap@1390}  size = 1
            > "getPrevious" -> {HashMap@1400}  size = 1
            ∨ "setDistance" -> {HashMap@1377}  size = 1
                > key = "setDistance"
                ∨ value = {HashMap@1377}  size = 1
                    ∨ "void" -> {ParameterList@1375}
                        > key = "void"
                        ∨ value = {ParameterList@1375}
                            ∨ parameterList = {ArrayList@1456}  size = 1
                                ∨ 0 = {Parameter@1458}
                                    > type = "int"
                                    > identifier = "distance"
```

TypeCheck

```
RefType.typeCheck {
        Einmaligkeit Feldbezeichner
        Each FieldDecl: typeCheck
        Einmaligkeit Methoden
        Each MethodDecl: typeCheck
}
```

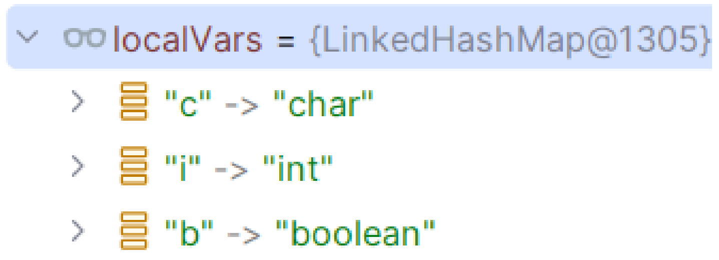
```
MethodDecl.typeCheck {
        Parameter zu localVars hinzufügen
        BlockStatement.typeCheck
}
```

```
BlockStatement.typeCheck {
        Each statement: typeCheck
        Aktualisierung localVars
        Rückgabetyp aller Pfade
}
```

IStatement, IExpression, IStatementExpression, IDatatype

<u>Lokale Variablen</u>

Varibalenname → Typbezeichnung



```
∨ oo localVars = {LinkedHashMap@1305}
    > "c" -> "char"
    > "i" -> "int"
    > "b" -> "boolean"
```

Bytecodgenerierung

- Startpunkt im AST: Program
  - codeGen- Methoden werden auf untergeordnete Elemente aufgerufen

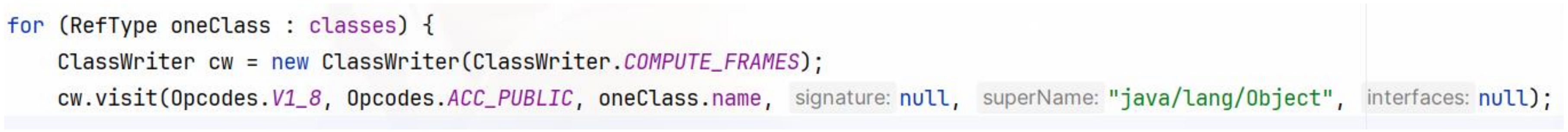
- Untergeordnete Elemente erben von Interfaces
  - IExpression, IStatement, IDatatype
  - Jedes Interface implementiert CodeGen()
  - Übergeben von LokalenVaraiblen, Klassen, Methoden

- Verwendung von Hilfsklassen und Typecheck
  - CodeGenHelper: LocalVar-Index, FieldDescriptor
  - Verwendung von GetTypeCheckResult() für Typ

- Exceptions bei Fehlern (Variable nicht gefunden, …)

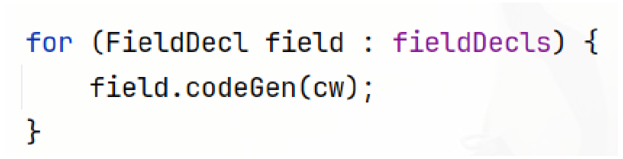
Bytecodgenerierung

- Program:

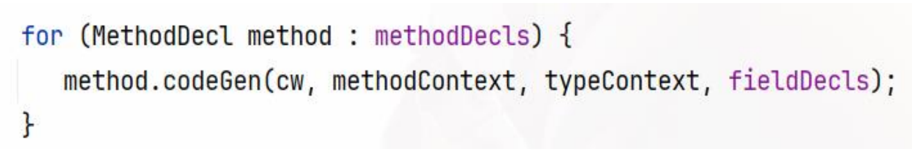


```
for (RefType oneClass : classes) {
    ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_FRAMES);
    cw.visit(Opcodes.V1_8, Opcodes.ACC_PUBLIC, oneClass.name,  signature: null,  superName: "java/lang/Object",  interfaces: null);
```

- RefType

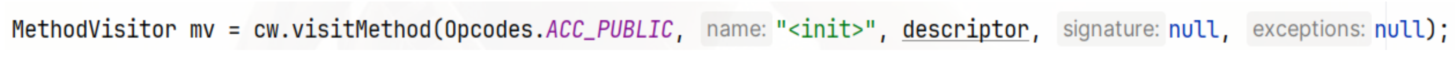


```
for (FieldDecl field : fieldDecls) {
    field.codeGen(cw);
}
```

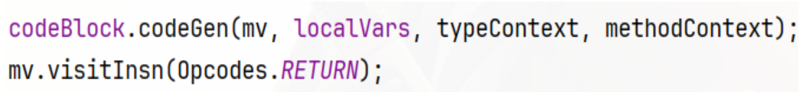


```
for (MethodDecl method : methodDecls) {
    method.codeGen(cw, methodContext, typeContext, fieldDecls);
}
```

- MethodDecl

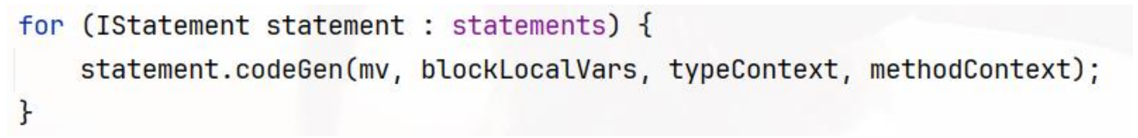


```
MethodVisitor mv = cw.visitMethod(Opcodes.ACC_PUBLIC,  name: "<init>", descriptor,  signature: null,  exceptions: null);
```



```
codeBlock.codeGen(mv, localVars, typeContext, methodContext);
mv.visitInsn(Opcodes.RETURN);
```

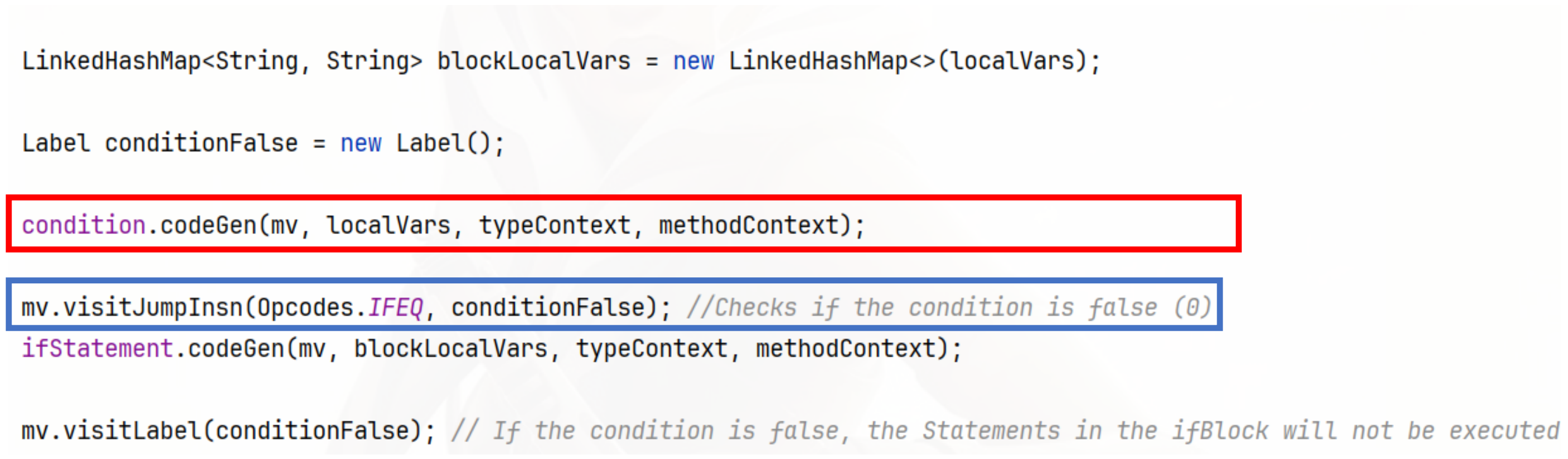
- BlockStatement



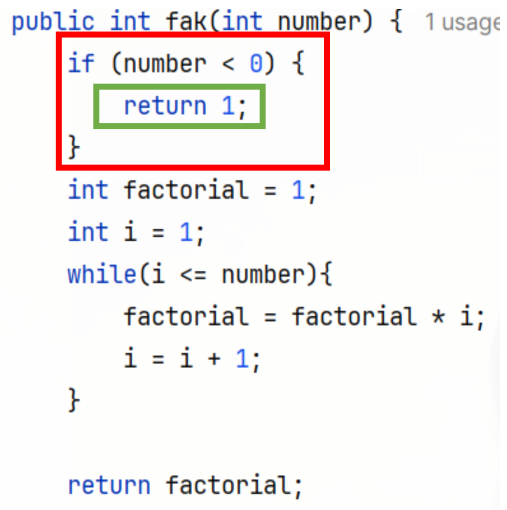
```
for (IStatement statement : statements) {
    statement.codeGen(mv, blockLocalVars, typeContext, methodContext);
}
```

$\longrightarrow$ IStatement, IExpression, StatementExpression, IDatatype

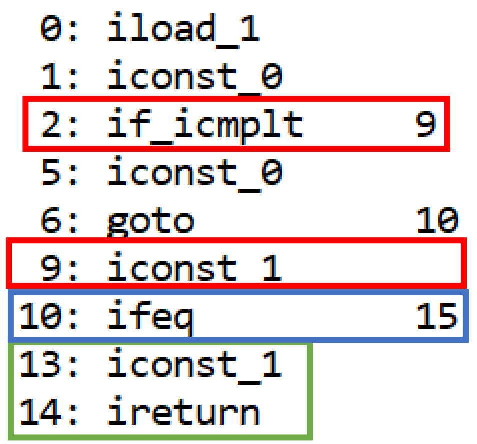
Bytecodgenerierung



```
LinkedHashMap<String, String> blockLocalVars = new LinkedHashMap<>(localVars);

Label conditionFalse = new Label();

condition.codeGen(mv, localVars, typeContext, methodContext);

mv.visitJumpInsn(Opcodes.IFEQ, conditionFalse); //Checks if the condition is false (0)
ifStatement.codeGen(mv, blockLocalVars, typeContext, methodContext);

mv.visitLabel(conditionFalse); // If the condition is false, the Statements in the ifBlock will not be executed
```



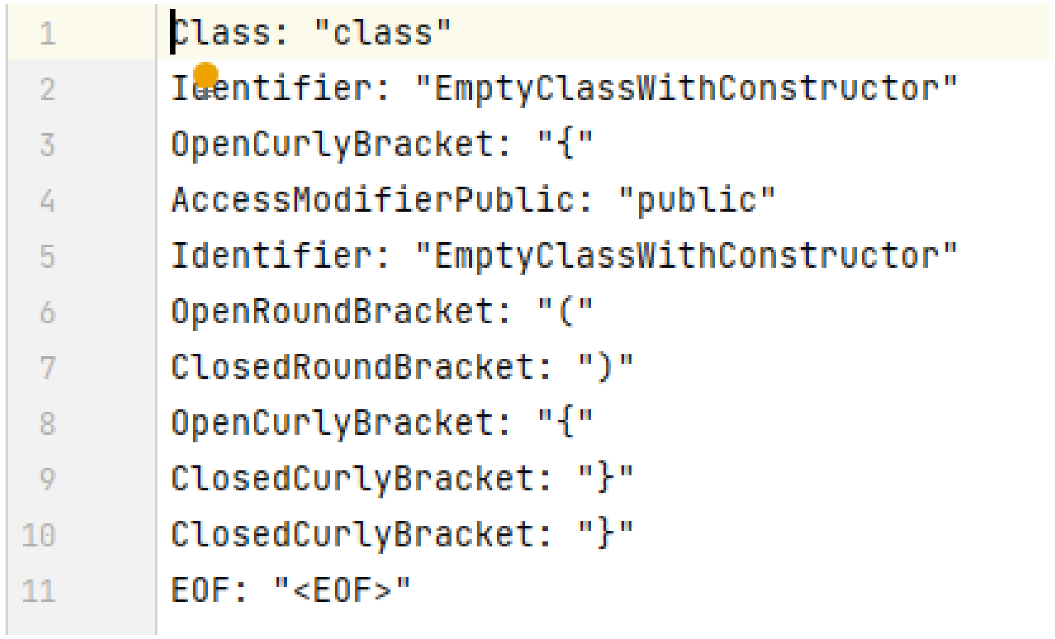
```
public int fak(int number) {  1 usage
    if (number < 0) {
        return 1;
    }
    int factorial = 1;
    int i = 1;
    while(i <= number){
        factorial = factorial * i;
        i = i + 1;
    }

    return factorial;
```



```
 0: iload_1
 1: iconst_0
 2: if_icmplt      9
 5: iconst_0
 6: goto          10
 9: iconst_1
10: ifeq          15
13: iconst_1
14: ireturn
```

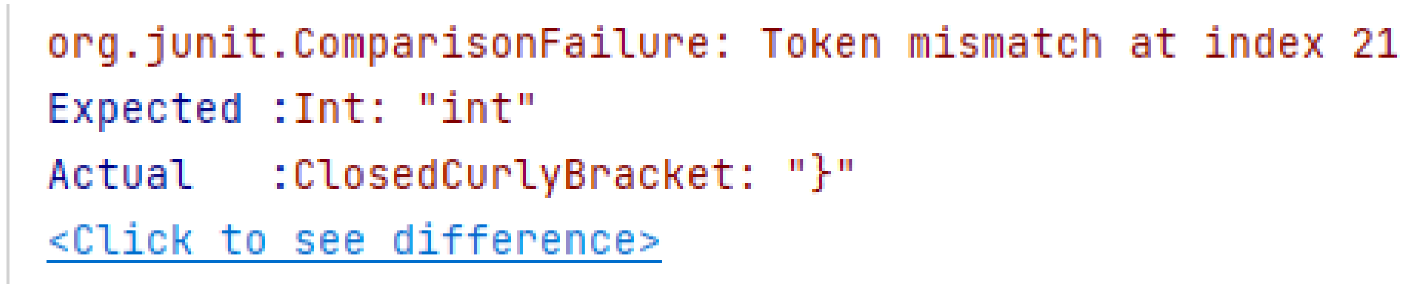
$\longrightarrow$ Vergleich mit 0

Token-Test



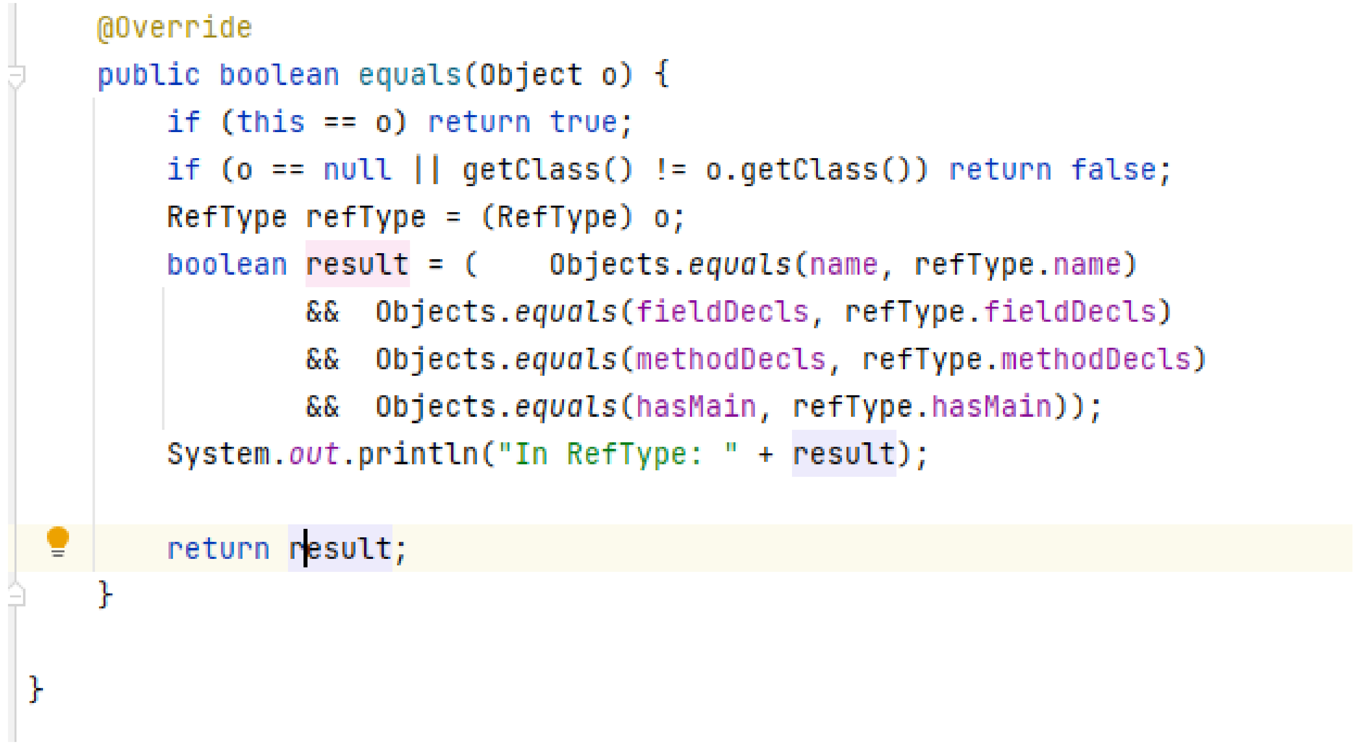
```
Class: "class"
Identifier: "EmptyClassWithConstructor"
OpenCurlyBracket: "{"
AccessModifierPublic: "public"
Identifier: "EmptyClassWithConstructor"
OpenRoundBracket: "("
ClosedRoundBracket: ")"
OpenCurlyBracket: "{"
ClosedCurlyBracket: "}"
ClosedCurlyBracket: "}"
EOF: "<EOF>"
```

Token-Test



```
org.junit.ComparisonFailure: Token mismatch at index 21
Expected :Int: "int"
Actual   :ClosedCurlyBracket: "}"
<Click to see difference>
```

AST-Test



```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        RefType refType = (RefType) o;
        boolean result = (    Objects.equals(name, refType.name)
                && Objects.equals(fieldDecls, refType.fieldDecls)
                && Objects.equals(methodDecls, refType.methodDecls)
                && Objects.equals(hasMain, refType.hasMain));
        System.out.println("In RefType: " + result);

        return result;
    }

}
```

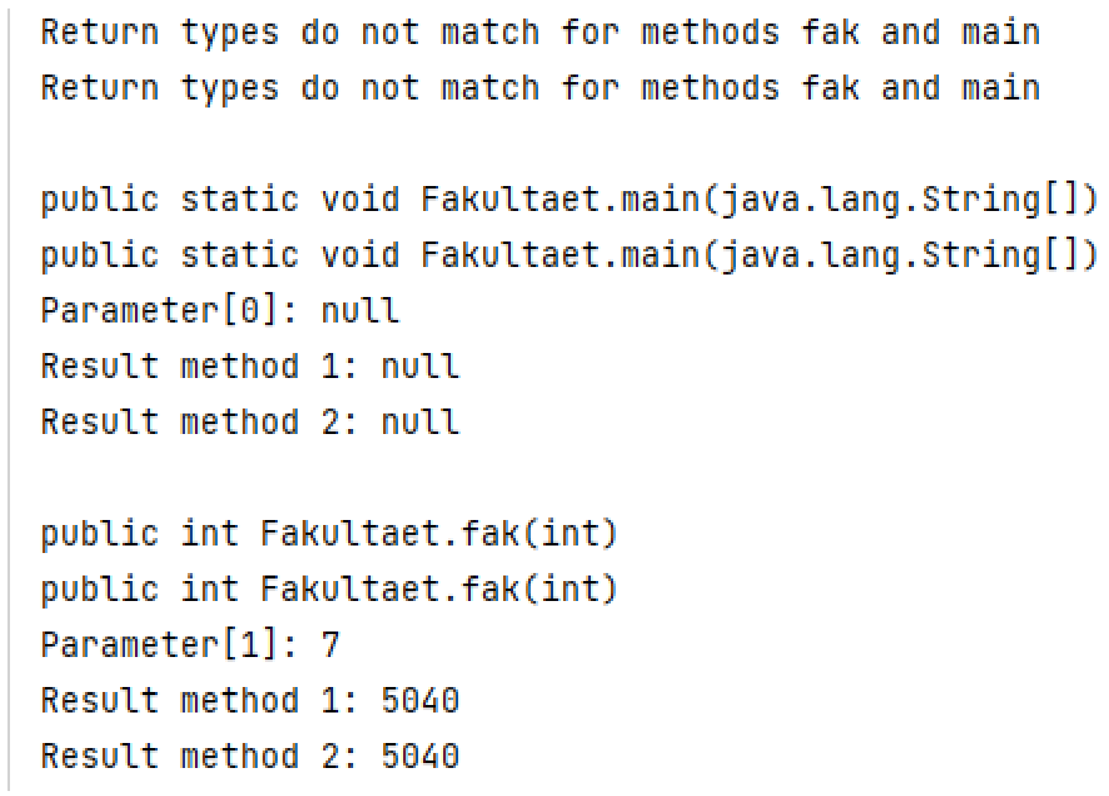
Typecheck-Test

- Läuft Typecheck durch ?
- TypeContext und MethodContext korrekt?
- Stimmt AST nach Veränderungen überein?

ByteCode-Test

- Vergleichsdatei erstellt mit Javac (und jar)
- Laden mit ClassFileLoader oder JarFileLoader
- ComparebyteCodeSyntax
- CompareByteCodeBehaviour

ByteCode-Test



```
Return types do not match for methods fak and main
Return types do not match for methods fak and main

public static void Fakultaet.main(java.lang.String[])
public static void Fakultaet.main(java.lang.String[])
Parameter[0]: null
Result method 1: null
Result method 2: null

public int Fakultaet.fak(int)
public int Fakultaet.fak(int)
Parameter[1]: 7
Result method 1: 5040
Result method 2: 5040
```