
DHBW
Stuttgart

Decaf Programmiersprache

- Simple Programmiersprache
- Java-like Syntax
- aber keine Klassen und Vererbung

Beispiel:

```
def int add(int x, int y)
{
    return x + y;
}

def int main()
{
    int a;
    a = 3;
    return add(a, 2);
}
```

Decaf: Syntax (1)

$$
\begin{aligned}
\mathit{Program} &\rightarrow (\mathit{Var} \mid \mathit{Method})* \\
\\
\mathit{Var} &\rightarrow \mathit{Type}\ \texttt{ID}\ \texttt{';'} \\
\mathit{Method} &\rightarrow \texttt{'def'}\ \mathit{Type}\ \texttt{ID}\ \texttt{'('}\ \mathit{Params}?\ \texttt{')'}\ \mathit{Block} \\
\mathit{Type} &\rightarrow \texttt{'int'} \mid \texttt{'bool'} \mid \texttt{'void'} \\
\\
\mathit{Block} &\rightarrow \texttt{'\{'}\ \mathit{Var}*\ \mathit{Stmt}*\ \texttt{'\}'} \\
\mathit{Stmt} &\rightarrow \mathit{Loc}\ \texttt{'='}\ \mathit{Expr}\ \texttt{';'} \\
&\quad \mid \mathit{MethodCall}\ \texttt{';'} \\
&\quad \mid \texttt{'if'}\ \texttt{'('}\ \mathit{Expr}\ \texttt{')'}\ \mathit{Block}\ (\texttt{'else'}\ \mathit{Block})? \\
&\quad \mid \texttt{'while'}\ \texttt{'('}\ \mathit{Expr}\ \texttt{')'}\ \mathit{Block} \\
&\quad \mid \texttt{'return'}\ \mathit{Expr}?\ \texttt{';'}
\end{aligned}
$$

Decaf: Syntax (2)

$$
\begin{aligned}
\mathit{Expr} &\rightarrow \mathit{Expr}\ \mathit{Operator}\ \mathit{Expr} \\
&\mid\ \texttt{'('}\ \mathit{Expr}\ \texttt{')'} \\
&\mid\ \mathit{Loc} \\
&\mid\ \mathit{MethodCall} \\
&\mid\ \mathit{Constant} \\
\mathit{Loc} &\rightarrow \texttt{ID} \\
\\
\mathit{MethodCall} &\rightarrow \texttt{ID}\ \texttt{'('}\ \mathit{Args}?\ \texttt{')'} \\
\mathit{Args} &\rightarrow \mathit{Expr}\ (\texttt{','}\ \mathit{Expr})* \\
\\
\mathit{Constant} &\rightarrow \texttt{NUMBER} \mid \texttt{'true'} \mid \texttt{'false'} \\
\mathit{Operator} &\rightarrow \texttt{'+'} \mid \texttt{'-'} \mid \texttt{'*'}
\end{aligned}
$$

Decaf: Abstrakte Syntax (1)

```
Program                     (enthält Variablen und Funktionen)
Variable
Function                    (enthält einen Block)
Block                       (enthält Variablen und Statements)
Statement
    Assignment              (Location = Expression)
    VoidFunctionCall        (wrapper für FunctionCall)
    IfElse                  (eine Expression und zwei Blöcke)
    WhileLoop               (enthält Expression und Block)
    Return                  (enthält Expression)
    VoidReturn              (return ohne Expression)
```

Decaf: Abstrakte Syntax (2)

```
Expression
    BinaryExpr                  (enthält zwei Expressions)
    Location                    (Variablenzugriff)
    FunctionCall                (enthält mehrere Expressions)
    Literal
        StringLiteral
        IntLiteral
        BoolLiteral
```

AST Implementierung (1)

- Implementierng als `record` Typen
  - Immutable (unveränderliche) Datenstruktur
  - Erstellt `get` und `set` Methoden automatisch

```
record Program(List<Variable> variablen, List<Function> funktionen) {}
record Variable(String name, Type type) {}
record Function(Type type, String name, List<Variable> params, Block block)

interface Expression {}
record BinaryExpr(Expression left, Operator op, Expression right) implements Expression {}
enum Operator { ADD, SUB, MUL }
record Location(String varName) implements Expression {}
record FunctionCall(String varName, List<Expression> params) implements Expression {}
record StringLiteral(String value) implements Expression {}
record IntLiteral(Integer value) implements Expression {}
record BoolLiteral(Boolean value) implements Expression {}
```

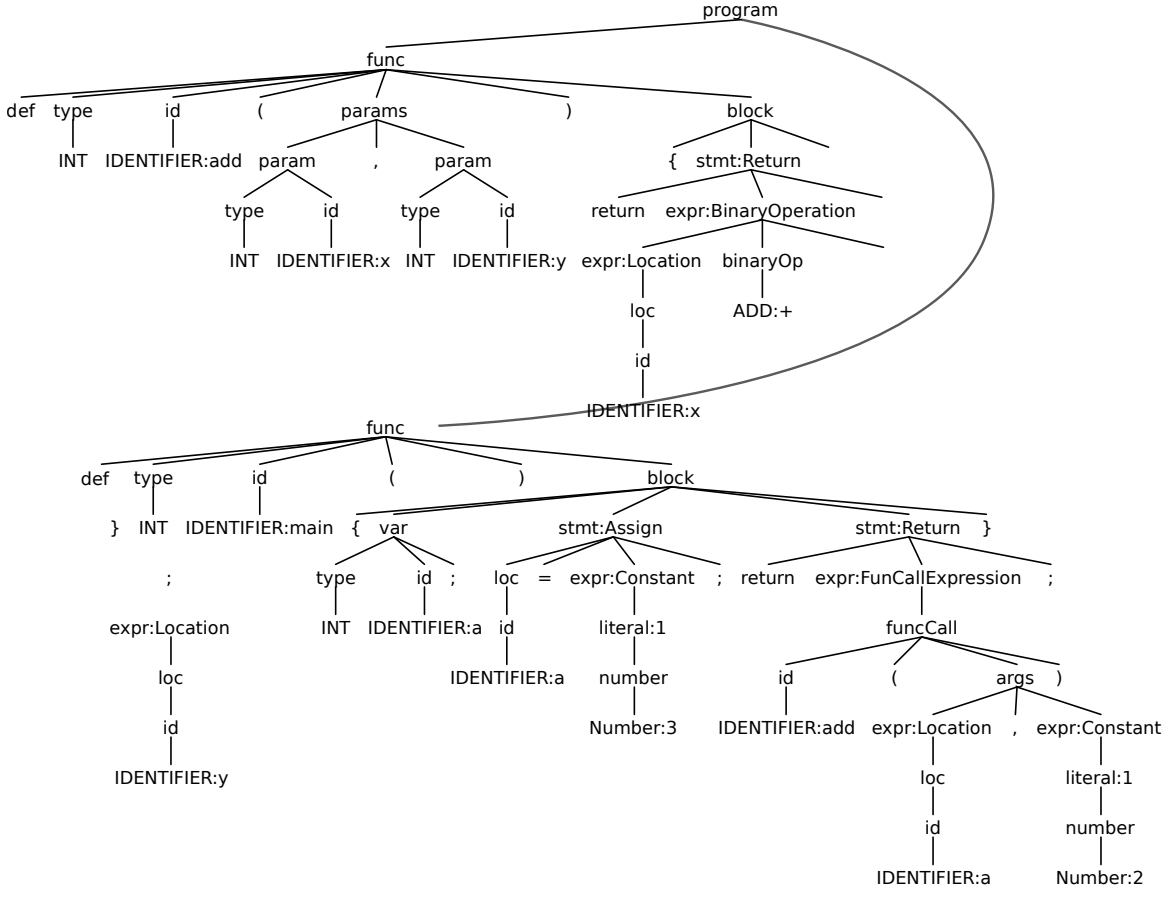
AST Implementierung (2)

- Statements:

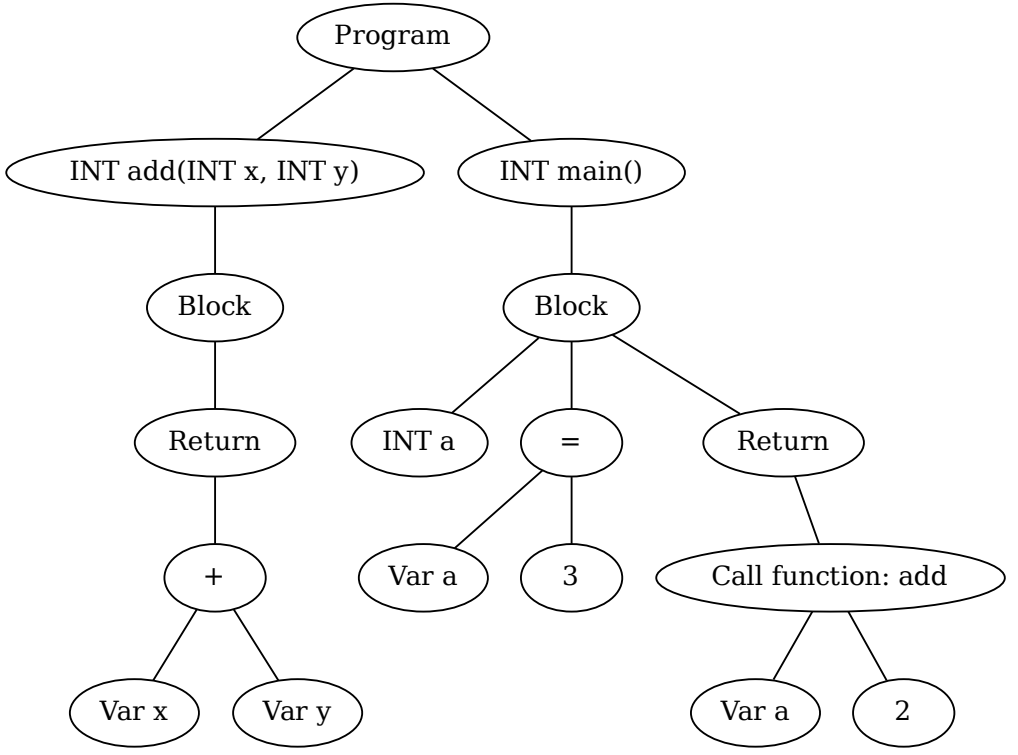
```
interface Statement {}
record Assignment(Location loc, Expression value) implements Statement {}
record VoidFunctionCall(FunctionCall expr) implements Statement {}
record ReturnVoid() implements Statement {}
record Return(Expression expr) implements Statement {}
record IfElse(Expression cond, Block ifBlock, Block elseBlock) implements Statement{}
record While(Expression cond, Block block) implements Statement {}
```

Umwandlung: ParseTree → AST

Parse Tree



Abstract Syntax Tree



ASTVisitor

- ANTLR erzeugt eine `Vistor` und eine `BaseVisitor` Klasse.
- Beispiel:

```
grammar Beispiel;

regel : unterRegel1 #Regel1
      | unterRegel2 #Regel2
      ;
unterRegel1 : "Hallo";
unterRegel2 : "Besucher";
```

generiert die `BeispielBaseVisitor` Klasse:

```
interface TestVisitor<T> {
    <T> visitRegel1(Regel1Context ctx);
    <T> visitRegel2(Regel2Context ctx);
}
```

ASTVisitor

Problem: `StmtContext` im `ParseTree` kann verschiedene Statements darstellen.

Kann beispielsweise durch Typcasts gelöst werden:

```
//Methode soll aus dem ParseTree Element Statement ein AST Element generieren:
Statement generate(StatementContext ctx){
    if(ctx instanceof WhileContext){
        WhileContext wCtx = (WhileContext) ctx;
        return ... //Generate While
    }
    if(ctx instanceof ReturnContext){
    ... }
}
```

ASTVisitor

Problem: `StmtContext` im `ParseTree` kann verschiedene Statements darstellen.

Lösung: Mithilfe der `BaseVisitor` Klasse geht es auch ohne Casts:

```
class StatementGenerator extends DecafBaseVisitor<Statement> {
    Statement visitWhile(WhileContext ctx){
        ...
    }
    Statement visitReturn(ReturnContext ctx){
            ...
    }
}
```

ASTVisitor - Grammatik Labels

```
stmt : loc '=' expr ';'                                  #Assign
     | funcCall ';'                                      #FunctionCall
     | 'if' '(' expr ')' block ('else' block)?           #If
     | 'while' '(' expr ')' block                        #While
     | 'return' expr ';'                                 #Return
     | 'return' ';'                                      #ReturnVoid
     | 'break' ';'                                       #Break
     | 'continue' ';'                                    #Continue
     ;
```

- Zusätzliche Strukturierung durch `#Labels`
- ANTLR generiert einen zusätzlichen Visitor für jedes Label

Übungsblock 3

Übungsblatt 2: Aufgabe 1 und 2

- Link zum Übungsblatt: `http://www2.ba-horb.de/~stan/%C3%BCbung2.pdf`
- Link zum Maven-Download: `https://maven.apache.org/download.cgi`