Interpreterbau

Für eine domänenspezifische Sprache

Leonie Collmann, itemis AG

Inhalt

- Warum Interpreterbau?
- Compiler vs. Interpreter
- Statement vs. Expression
- Aufbau des Syntaxbaums
- Aufbau eines Modellierten Programms
- Bausteine des Interpreters
- Praxisbeispiel
- Übersicht

Motivation - Warum Interpreterbau?

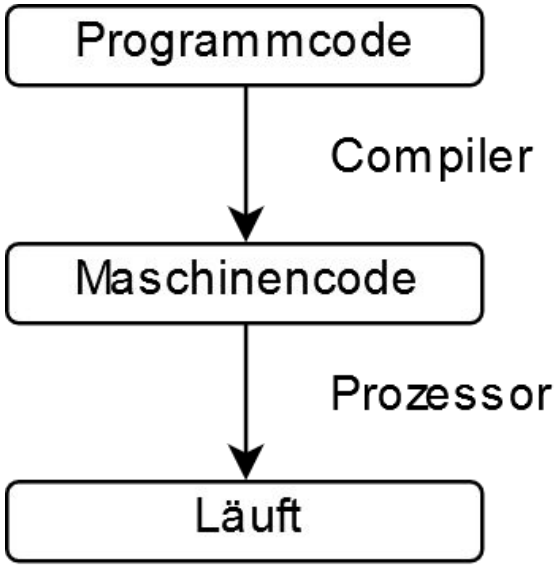
- Entwicklung einer eigenen Sprache, z.B. für Kundenprojekt

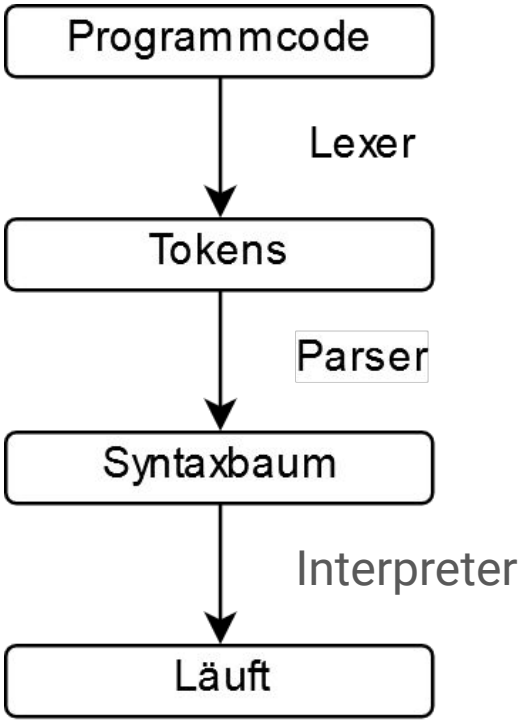
  -> Xtext für Grammatik und Parser, Interpreter in Eigenregie

- Maschinencode schreiben ist schwieriger!

  Einige Kunden wünschen eine auf ihre Bedürfnisse zugeschnittene Sprache, die z.B. einfach für Nicht-Programmierer zu schreiben ist. Wenn man sich mit Tools wie Xtext eine Grammatik und einen Parser dazu erstellt, kann man auch manuell einen Interpreter dazu entwickeln, sodass der geparste Code-Output sinnvoll ausgeführt werden kann.

Compiled vs. Interpreted



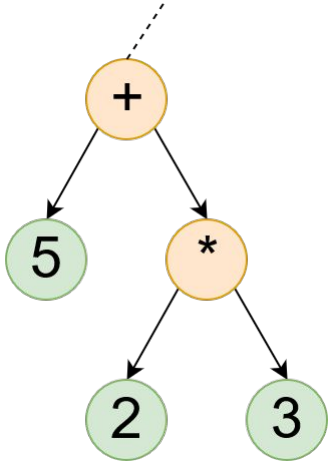


Unterschied Statement <> Expression

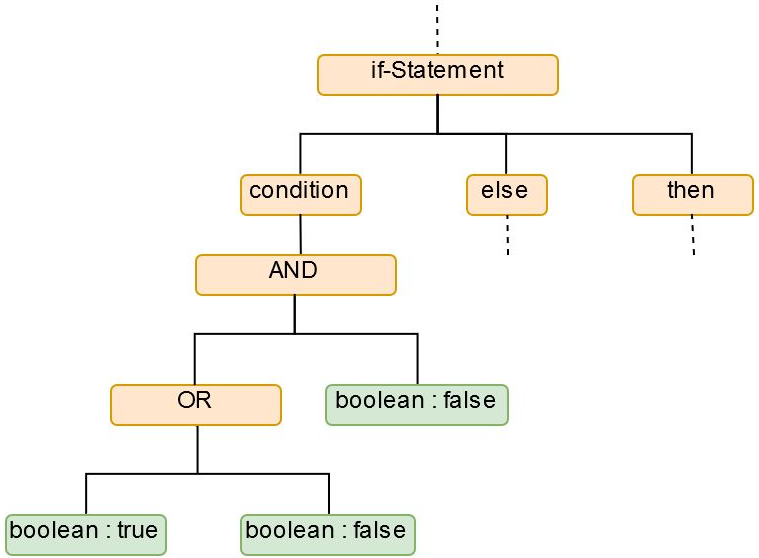
- Statement:
  - führt etwas aus
  - z.B. Loop, if-Statement, Variablenzuweisung

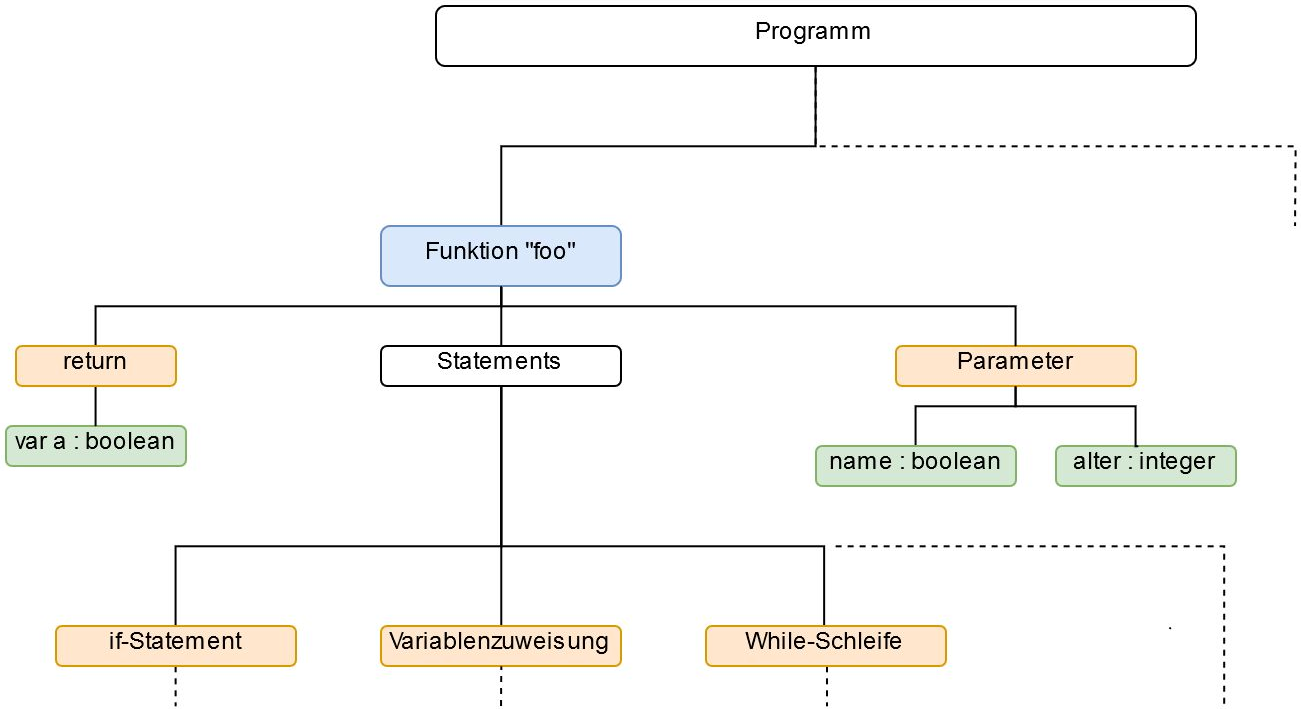
- Expression:
  - extends Statement
  - beinhaltet Wert, gibt Wert zurück
  - z.B. Variable, AND/OR-Expression

Syntaxbaum



- Syntaxbaum Basis für Interpretieren
- Gerüst aus Statements und Expressions
- (einfache) Expressions an den Blättern
- Syntaxbaum muss durch Parsen der Sprache erstellt werden (hier Blackbox)
- Interpretieren: Traversierung der Zweige

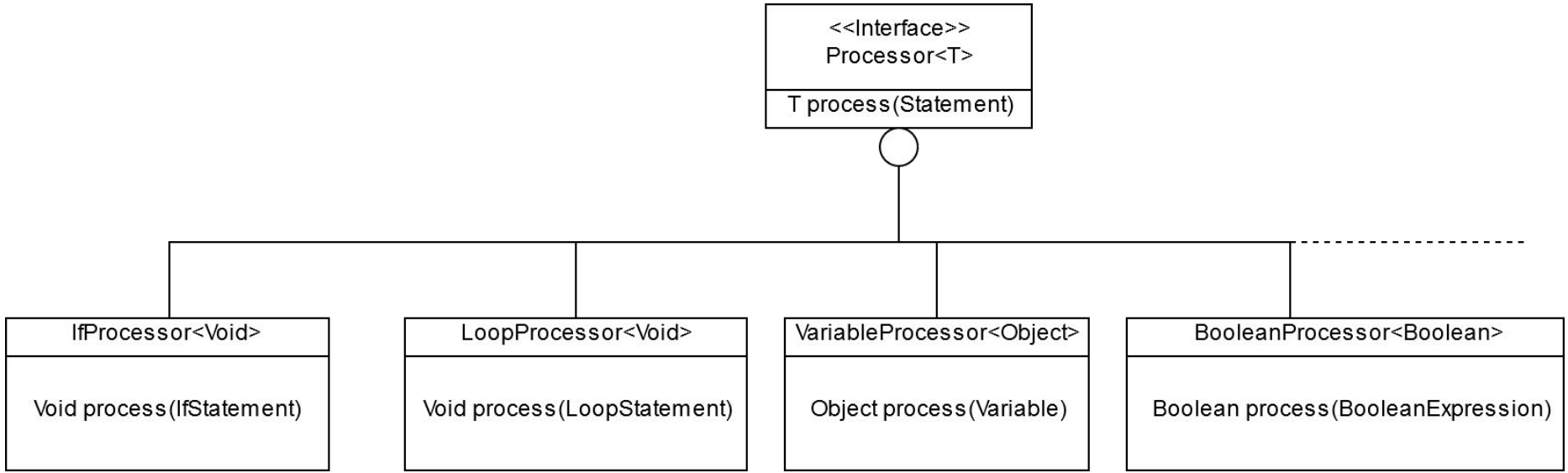

if-Statement
condition
else
then
AND
OR
boolean : false
boolean : true
boolean : false


Programm
Funktion "foo"
return
Statements
Parameter
var a : boolean
name : boolean
alter : integer
if-Statement
Variablenzuweisung
While-Schleife

Aufbau des Interpreters

- Hauptklasse Interpreter
- verschiedene Prozessor<Statement> mit Factory dazu
- VariablenManager
- Exceptions (z.B. WrongTypeException für nicht-passende Datentypen)

Processor für Statements



ProcessorFactory

Nimmt Statement, gibt neuen Processor zurück, der auf das Statement passt.

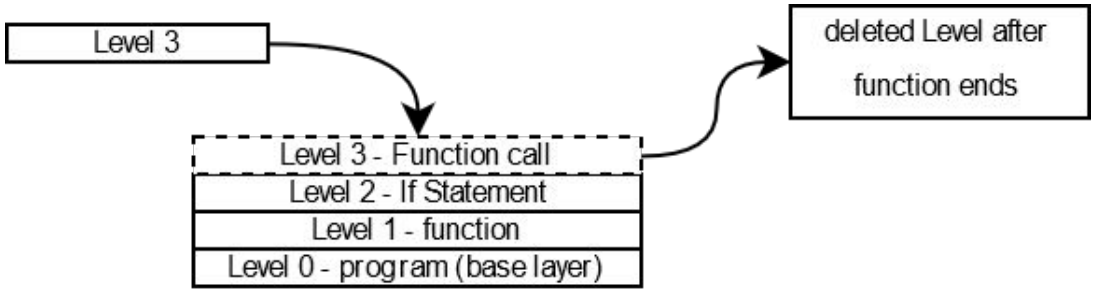
Processor<?> createProcessor(Statement s)

Implementiert z.B. mit

```
if(s instanceof IfStatement) {
    return new IfProcessor();
}
```

VariableManager (1)

- hat einen Stack aus Listen von Variablen für verschiedene Scope-Level



- VariableProcessor ruft add(Variable) im VarManager auf, sodass Variablen dem Stack hinzugefügt werden

VariableManager (2)

Ein VariablenManager hat z.B. folgende Funktionen:

add(Variable)

delete(Variable)

getValue(Variable)

contains(Variable)

addLevel() - bei Betreten einer Ebene neues Variablen-Level

deleteLevel() - nach Beenden einer Ebene werden die Variablen entfernt

Beispiel Interpreter im Code

In Eclipse anhand eines Dummy Interpreters für eine Dummy Sprache.

Es wurden beispielhaft Statements und Expressions modelliert, z.B.

ANDExpression, BooleanExpression, IfStatement

Und diese mit Hilfe von passenden Processoren ausgeführt.

Code zu finden auf https://github.com/leo-collmann/dummy-interpreter

Übersicht

- Statements und Expressions werden abgewandert
- Ergebnisse der Expressions werden an Eltern weitergegeben
- Prozessoren übernehmen die eigentliche Interpretationsarbeit
- Je nach Sprache sind viele weitere Bestandteile möglich/nötig
- Mini-Demo zu finden auf https://github.com/leo-collmann/dummy-interpreter

Fragen?