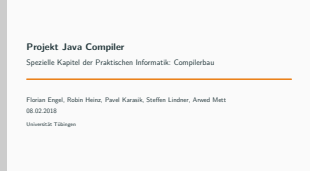
Projekt Java Compiler

Spezielle Kapitel der Praktischen Informatik: Compilerbau

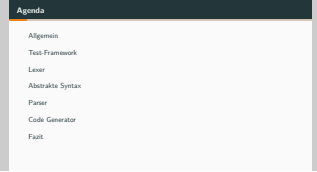
Florian Engel, Robin Heinz, Pavel Karasik, Steffen Lindner, Arwed Mett

08.02.2018

Universität Tübingen



Agenda



Allgemein

Allgemein

Allgemein

Aufgabenstellung:

Entwickeln eines Mini-Java Compilers mit den zugehörigen Schritten: Lexer, Parser, TypChecker und Codegenerierung.

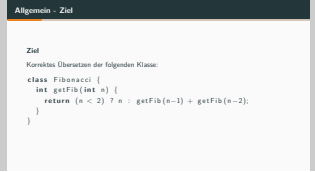
Allgemein
Aufgabenstellung:
Entwickeln eines Mini-Java Compilers mit den zugehörigen Schritten: Lexer, Parser, TypChecker und Codegenerierung.

Allgemein - Ziel

Ziel

Korrektes Übersetzen der folgenden Klasse:

```
class Fibonacci {
  int getFib(int n) {
    return (n < 2) ? n : getFib(n-1) + getFib(n-2);
  }
}
```

Allgemein - Featureliste

Umgesetze Features (Auszug):

Allgemein - Featureliste
Umgesetze Features (Auszug):

Allgemein - Featureliste

Umgesetze Features (Auszug):

- Ternary Operator

Allgemein - Featureliste
Umgesetze Features (Auszug):
- Ternary Operator

Allgemein - Featureliste

Umgesetze Features (Auszug):

- Ternary Operator
- For / While / DoWhile

Allgemein - Featureliste
Umgesetze Features (Auszug):
- Ternary Operator
- For / While / DoWhile

Allgemein - Featureliste

Umgesetze Features (Auszug):

- Ternary Operator
- For / While / DoWhile
- If / If-Else / Switch-Case

Allgemein - Featureliste

Umgesetze Features (Auszug):

- Ternary Operator
- For / While / DoWhile
- If / If-Else / Switch-Case
- Pre- bzw. Post Inkrement/Dekrement

Allgemein - Featureliste

Umgesetze Features (Auszug):

- Ternary Operator
- For / While / DoWhile
- If / If-Else / Switch-Case
- Pre- bzw. Post Inkrement/Dekrement
- Arithmetische Operatoren (+, -, /, div, mod, *) inklusive Zuweisung (+=, etc.)

Allgemein - Entwicklung

Code-Sharing über GitHub (https://github.com/Pfeifenjoy/compilerbau-WS17-18) mit continuous integration (travis).

Allgemein - Entwicklung
Code-Sharing über GitHub (https://github.com/Pfeifenjoy/compilerbau-WS17-18) mit continuous integration (travis).

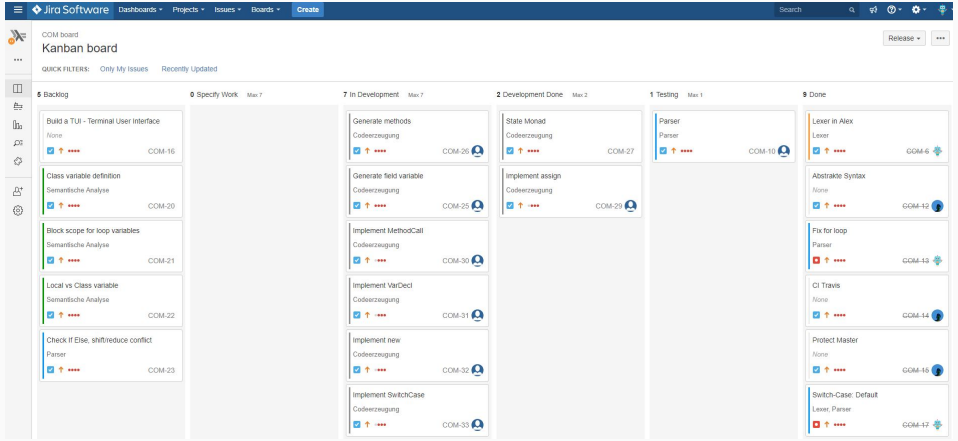
Allgemein - Entwicklung

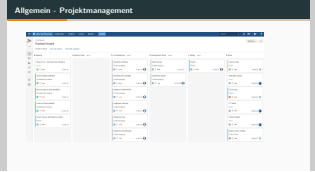
Code-Sharing über GitHub (https://github.com/Pfeifenjoy/compilerbau-WS17-18) mit continuous integration (travis).

Als Build-System wird cabal eingesetzt.

Allgemein - Entwicklung
Code-Sharing über GitHub (https://github.com/Pfeifenjoy/compilerbau-WS17-18) mit continuous integration (travis).
Als Build-System wird cabal eingesetzt.

Allgemein - Projektmanagement



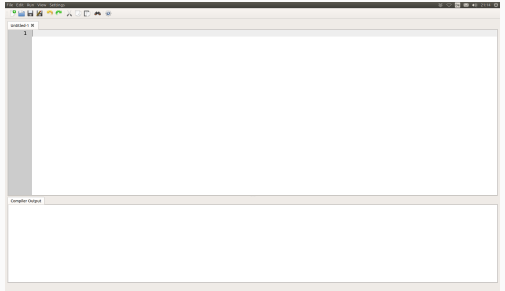


Allgemein - JC

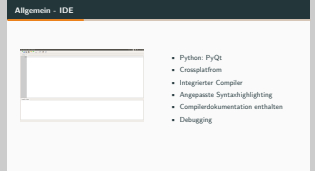
- Text User Interface
- Wechseln in Ordner: dist/build/jc
- Hilfe: ./jc -h
- Compile mit Log: ./jc File.java -l logFile

Allgemein - JC
- Text User Interface
- Wechseln in Ordner: dist/build/jc
- Hilfe: ./jc -h
- Compile mit Log: ./jc File.java -l logFile

Allgemein - IDE



- Python: PyQt
- Crossplatfrom
- Integrierter Compiler
- Angepasste Syntaxhighlighting
- Compilerdokumentation enthalten
- Debugging



Test-Framework

Test-Framework

Test-Framework - Allgemein

Das Test-Framework wurde selbst implementiert. Es enthält diverse Funktionen zum automatisierten Überprüfen der Testfälle.

Test-Framework - Allgemein
Das Test-Framework wurde selbst implementiert. Es enthält diverse Funktionen zum automatisierten Überprüfen der Testfälle.

Test-Framework - Allgemein

Das Test-Framework wurde selbst implementiert. Es enthält diverse Funktionen zum automatisierten Überprüfen der Testfälle.

Tests werden in korrekte und falsche Testfälle unterteilt.

Test-Framework - Token-Coverage & Testfälle

Die Test-Suite umfasst eine Token-Coverage von 100%.

Test-Framework - Token-Coverage & Testfälle

Die Test-Suite umfasst eine Token-Coverage von 100%.

Zusätzlich umfasst die Test-Suite insgesamt 21 gültige und 12 ungültige Testfälle.

Test-Framework - Token-Coverage & Testfälle

Die Test-Suite umfasst eine Token-Coverage von 100%.

Zusätzlich umfasst die Test-Suite insgesamt 21 gültige und 12 ungültige Testfälle.

Ungültige Testfälle können in Syntaxfehler (Parser) und Typfehler (Typchecker) eingeteilt werden.

Test-Framework - Testfälle

Jedes Testfile liegt in einem Ordner (Correct bzw. Wrong) mit zugehöriger .java-Datei.

Test-Framework - Testfälle
Jedes Testfile liegt in einem Ordner (Correct bzw. Wrong) mit zugehöriger .java-Datei.

Test-Framework - Testfälle

Jedes Testfile liegt in einem Ordner (Correct bzw. Wrong) mit zugehöriger .java-Datei.

Ein Testfile besteht aus:

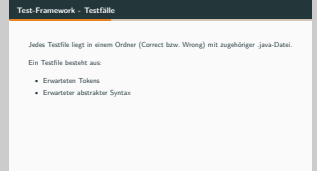
- Erwarteten Tokens

Test-Framework - Testfälle

Jedes Testfile liegt in einem Ordner (Correct bzw. Wrong) mit zugehöriger .java-Datei.

Ein Testfile besteht aus:

- Erwarteten Tokens
- Erwarteter abstrakter Syntax

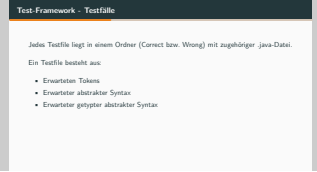


Test-Framework - Testfälle

Jedes Testfile liegt in einem Ordner (Correct bzw. Wrong) mit zugehöriger .java-Datei.

Ein Testfile besteht aus:

- Erwarteten Tokens
- Erwarteter abstrakter Syntax
- Erwarteter getypter abstrakter Syntax



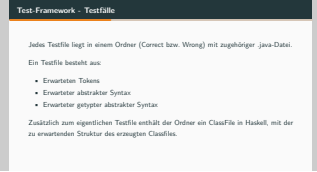
Test-Framework - Testfälle

Jedes Testfile liegt in einem Ordner (Correct bzw. Wrong) mit zugehöriger .java-Datei.

Ein Testfile besteht aus:

- Erwarteten Tokens
- Erwarteter abstrakter Syntax
- Erwarteter getypter abstrakter Syntax

Zusätzlich zum eigentlichen Testfile enthält der Ordner ein ClassFile in Haskell, mit der zu erwartenden Struktur des erzeugten Classfiles.



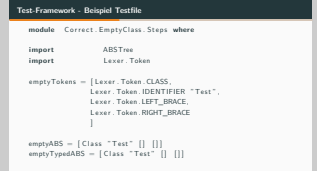
Test-Framework - Beispiel Testfile

```
module Correct.EmptyClass.Steps where

import          ABSTree
import          Lexer.Token

emptyTokens = [Lexer.Token.CLASS,
               Lexer.Token.IDENTIFIER "Test",
               Lexer.Token.LEFT_BRACE,
               Lexer.Token.RIGHT_BRACE
               ]

emptyABS = [Class "Test" [] []]
emptyTypedABS = [Class "Test" [] []]
```

Test-Framework - Beispielprogramme

Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.

Test-Framework - Beispielprogramme
Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.

Test-Framework - Beispielprogramme

Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.

- Multiplikation

Test-Framework - Beispielprogramme
Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.
- Multiplikation

Test-Framework - Beispielprogramme

Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.

- Multiplikation
- Gaußsumme (kleiner Gauß)

Test-Framework - Beispielprogramme
Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.
- Multiplikation
- Gaußsumme (kleiner Gauß)

Test-Framework - Beispielprogramme

Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.

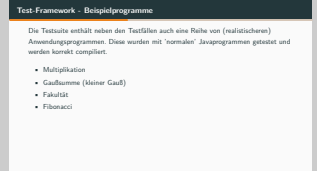
- Multiplikation
- Gaußsumme (kleiner Gauß)
- Fakultät

Test-Framework - Beispielprogramme
Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.
- Multiplikation
- Gaußsumme (kleiner Gauß)
- Fakultät

Test-Framework - Beispielprogramme

Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.

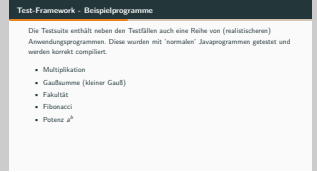
- Multiplikation
- Gaußsumme (kleiner Gauß)
- Fakultät
- Fibonacci



Test-Framework - Beispielprogramme

Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.

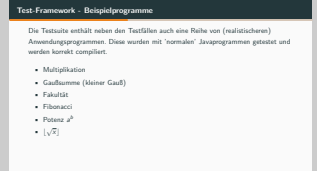
- Multiplikation
- Gaußsumme (kleiner Gauß)
- Fakultät
- Fibonacci
- Potenz a^b



Test-Framework - Beispielprogramme

Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.

- Multiplikation
- Gaußsumme (kleiner Gauß)
- Fakultät
- Fibonacci
- Potenz a^b
- $\lfloor \sqrt{x} \rfloor$



Test-Framework - Beispielprogramme

Die Testsuite enthält neben den Testfällen auch eine Reihe von (realistischeren) Anwendungsprogrammen. Diese wurden mit 'normalen' Javaprogrammen getestet und werden korrekt compiliert.

- Multiplikation
- Gaußsumme (kleiner Gauß)
- Fakultät
- Fibonacci
- Potenz a^b
- $\lfloor\sqrt{x}\rfloor$
- Primzahltest & nächste Primzahl ermitteln

Demo

Test-Framework: Demo
Demo

Lexer

Lexer

Lexer - Tokens

```
data Token
        -- Arithmetics
        = ADD
        | SUBTRACT
        | MULTIPLY
        | DIVIDE
        | MODULO
        | INCREMENT
        | DECREMENT
        -- Logical
        | NOT
        ...
```


Lexer - Struktur Alex File

```
%wrapper "basic"

$digit = 0-9
$alpha = [a-zA-Z]

tokens :-

        -- Ignore
        $white+ ;
        "//".*  ;

        -- Operators
        -- Arithmetics
        \+                  { \s -> ADD }
        \-                  { \s -> SUBTRACT }
        \*                  { \s -> MULTIPLY }
        \/                  { \s -> DIVIDE }
        ...
```


Demo

Lexer - Demo
Demo

Abstrakte Syntax

Abstrakte Syntax

ABSTree.hs

Abstrakte Syntax
ABSTree.hs

Parser

Parser

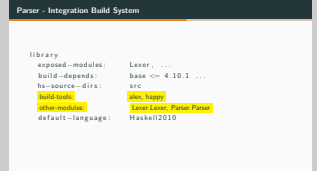
Parser - Allgemein

- Verwendete Werkzeuge: Happy, Alex
- Input: Tokens
- Output: ABSTree
- Herausforderungen:
  - Integration in Build System
  - Operatoren Priorität
  - Klassen / Konstruktoren
  - If-Else

Parser - Allgemein
- Verwendete Werkzeuge: Happy, Alex
- Input: Tokens
- Output: ABSTree
- Herausforderungen:
  - Integration in Build System
  - Operatoren Priorität
  - Klassen / Konstruktoren
  - If-Else

Parser - Integration Build System

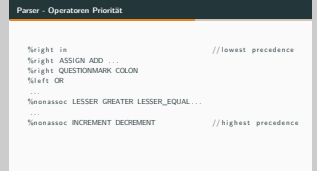
```
library
  exposed-modules:      Lexer, ...
  build-depends:        base <= 4.10.1 ...
  hs-source-dirs:       src
  build-tools:          alex, happy
  other-modules:        Lexer.Lexer, Parser.Parser
  default-language:     Haskell2010
```

- build-tools
- other-modules

Parser - Operatoren Priorität

```
%right in                                   //lowest precedence
%right ASSIGN ADD ...
%right QUESTIONMARK COLON
%left OR
 ...
%nonassoc LESSER GREATER LESSER_EQUAL...
 ...
%nonassoc INCREMENT DECREMENT               //highest precedence
```

- Was ist Operatoren Priorität
- Wie wird es in Happy implementiert
- Precedence top (low), bottom (high)
- Beschreibe Assoziativität

Parser - Struktur Happy File

```
Program
    : Class                 { [$1] }
    | Program Class         { $1 ++ [$2] }
    | Program SEMICOLON     { $1 }

Statement
    : SingleStatement SEMICOLON             { $1 }
      ...
    | IF LEFT_PARANTHESES Expression RIGHT_PARANTHESES
      Statement ELSE Statement
                                            { If $3 $5 (Just $7) }
    | IF LEFT_PARANTHESES Expression
      RIGHT_PARANTHESES Statement
      %prec THEN                            { If $3 $5 Nothing }
    | Switch                                { $1 }
```


Parser - Klassen / Konstruktoren

Problem:

- Methoden, Attribute aufteilen in Listen
- teste: Klassenname = Konstruktorname

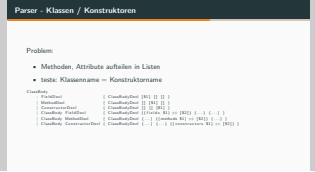
Parser - Klassen / Konstruktoren
Problem:
- Methoden, Attribute aufteilen in Listen
- teste: Klassenname = Konstruktorname

Parser - Klassen / Konstruktoren

Problem:

- Methoden, Attribute aufteilen in Listen
- teste: Klassenname = Konstruktorname

```
ClassBody
    : FieldDecl                 { ClassBodyDecl [$1] [] [] }
    | MethodDecl                { ClassBodyDecl [] [$1] [] }
    | ConstructorDecl           { ClassBodyDecl [] [] [$1] }
    | ClassBody FieldDecl       { ClassBodyDecl ((fields $1) ++ [$2]) (...) (...) }
    | ClassBody MethodDecl      { ClassBodyDecl (...) ((methods $1) ++ [$2]) (...) }
    | ClassBody ConstructorDecl { ClassBodyDecl (...) (...) ((constructors $1) ++ [$2]) }
```

Parser - If - Else

Problem:

```
if_stmt
    : "if" expr "{" stmt "}"
    | "if" expr "{" stmt "}" "else" "{" statement "}"
```


Parser - If - Else

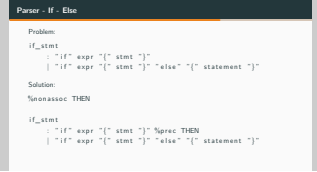
Problem:

```
if_stmt
    : "if" expr "{" stmt "}"
    | "if" expr "{" stmt "}" "else" "{" statement "}"
```

Solution:

```
%nonassoc THEN

if_stmt
    : "if" expr "{" stmt "}" %prec THEN
    | "if" expr "{" stmt "}" "else" "{" statement "}"
```

Demo

Beispiel
Demo

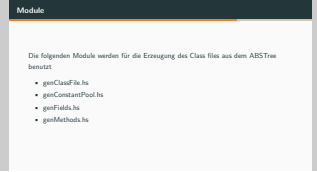
Code Generator

Code Generator

Module

Die folgenden Module werden für die Erzeugung des Class files aus dem ABSTree benutzt

- genClassFile.hs
- genConstantPool.hs
- genFields.hs
- genMethods.hs



Module

In den nachfolgenden Modulen sind die Datentypen enthalten die für den abstrakten Bytecode benutzt werden

- Data/Assembler.hs
- Data/ClassFile.hs

Aus dem abstrakten Bytecode wird im Module Module BinaryClass.hs der Bytecode erzeugt

Module

In den nachfolgenden Modulen sind die Datentypen enthalten die für den abstrakten Bytecode benutzt werden

- Data/Assembler.hs
- Data/ClassFile.hs

Aus dem abstrakten Bytecode wird im Module Module BinaryClass.hs der Bytecode erzeugt

Constanten Pool

Der Constantenpool ist in einer hashMap die ein Eintrag auf dessen Position abbildet.

Im Module genConstantPool.hs sind Funktionen enthalten die ein Eintrag erzeugen und dessen Position zurückgeben bzw nur die Position zurückgeben.

Constanten Pool
Der Constantenpool ist in einer hashMap die ein Eintrag auf dessen Position abbildet.
Im Module genConstantPool.hs sind Funktionen enthalten die ein Eintrag erzeugen und dessen Position zurückgeben bzw nur die Position zurückgeben.

Beispiel genConstantPool

```
genMethodRefSuper :: String
                  -> Type
                  -> State ClassFile IndexConstantPool
genMethodRefSuper name typ =
  do indexClassName <- view (super . indexSp) <$> get
     indexNameType <- genNameAndType name typ
     genInfo MethodRefInfo
               { _tagCp             = TagMethodRef
               , _indexNameCp       = indexClassName
               , _indexNameandtypeCp = indexNameType
               , _desc              = ""
               }
```


Generieren der Methoden

Bei der Generierung der Methoden werden auch gleichzeitig die Einträge im constanten pool erstellt. Im State wird folgender Datentyp verwendet.

```
data Vars
  = Vars { _localVar :: [HM.HashMap LocVarName LocVarIndex]
         , _allLocalVar :: S.Set LocVarName
         , _classFile :: ClassFile
         , _curStack :: Int
         , _maxStack :: Int
         , _line :: LineNumber
         , _continueLine :: [LineNumber]
         }
makeLenses ''Vars
```


Generieren der Methoden
Bei der Generierung der Methoden werden auch gleichzeitig die Einträge im constanten pool erstellt. Im State wird folgender Datentyp verwendet.
data Vars
= Vars { _localVar :: [HM.HashMap LocVarName LocVarIndex]
, _allLocalVar :: S.Set LocVarName
, _classFile :: ClassFile
, _curStack :: Int
, _maxStack :: Int
, _line :: LineNumber
, _continueLine :: [LineNumber]
}
makeLenses ''Vars

Als Major Version wird 48 anstatt 53 verwendet da die StackMapTable nicht implemntiert wurde

Als Major Version wird 48 anstatt 53 verwendet da die StackMapTable nicht implemntiert wurde

Fazit

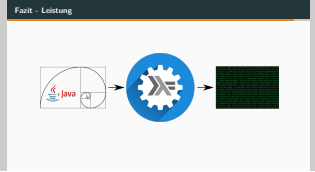
Fazit

Fazit - Probleme

- Kommunikation
- Absprung von Kommilitonen
- Projektmanagement

Fazit - Probleme
- Kommunikation
- Absprung von Kommilitonen
- Projektmanagement

Fazit - Leistung



Fazit - Matrix

Programm	Lexer	Parser	Typchecker	Codegenerator
Multiplikation	✓	✓	✓	✓
Gaußsumme (kleiner Gauß)	✓	✓	✓	✓
Fakultät	✓	✓	✓	✓
Fibonacci	✓	✓	✓	✓
Potenz a^b	✓	✓	✓	✓
Primzahlen	✓	✓	✓	✓

Fazit - Matrix

Programm	Lexer	Parser	Typchecker	Codegenerator
Multiplikation	✓	✓	✓	✓
Gaußsumme (kleiner Gauß)	✓	✓	✓	✓
Fakultät	✓	✓	✓	✓
Fibonacci	✓	✓	✓	✓
Potenz a^b	✓	✓	✓	✓
Primzahlen	✓	✓	✓	✓

Vielen Dank für Ihre Aufmerksamkeit

Fazit - Vielen Dank
Vielen Dank für Ihre Aufmerksamkeit

Projekt Java Compiler

Spezielle Kapitel der Praktischen Informatik: Compilerbau

Github: https://github.com/Pfeifenjoy/compilerbau-WS17-18

Florian Engel, florian.engel@student.uni-tuebingen.de

Robin Heinz, robin.heinz@student.uni-tuebingen.de

Pavel Karasik, pavel.karasik@student.uni-tuebingen.de

Steffen Lindner, steffen.lindner@student.uni-tuebingen.de

Arwed Mett, arwed.mett@student.uni-tuebingen.de

Projekt Java Compiler
Spezielle Kapitel der Praktischen Informatik: Compilerbau
Github: https://github.com/Pfeifenjoy/compilerbau-WS17-18

Florian Engel, florian.engel@student.uni-tuebingen.de
Robin Heinz, robin.heinz@student.uni-tuebingen.de
Pavel Karasik, pavel.karasik@student.uni-tuebingen.de
Steffen Lindner, steffen.lindner@student.uni-tuebingen.de
Arwed Mett, arwed.mett@student.uni-tuebingen.de