Bytecode

Contents

[Befehle 1](#_Toc66808902)

[Lesen und Schreiben von Variablen 1](#_Toc66808903)

[Arithmetische Berechnungen 2](#_Toc66808904)

[Sprungbefehle: 1 Parameter 2](#_Toc66808905)

[Sprungbefehle: compare two parameters 2](#_Toc66808906)

[Sprungbefehle: nullwerte 2](#_Toc66808907)

[Sprungbefehle: return 2](#_Toc66808908)

[FAQ 3](#_Toc66808909)

[Funktionen 3](#_Toc66808910)

[Wie kann man eine Funktion this.f aufrufen? 3](#_Toc66808911)

[Aufruf mit parameter? 3](#_Toc66808912)

[Aufruf mit parameter, der inline berechnet wird? 3](#_Toc66808913)

[Muss man ein „this“ hinzufügen, wenn es nicht explizit steht? 4](#_Toc66808914)

[IFs 4](#_Toc66808915)

[In welcher Rheihenfolge müssen Variablen auf Stack liegen? 4](#_Toc66808916)

[Code generation 4](#_Toc66808917)

[Minimale vs nichtminimale Stackverbrauch 4](#_Toc66808918)

[Code 5](#_Toc66808919)

# Befehle

## Lesen und Schreiben von Variablen

x - Nummer von Variable a

?- steht für typ. i - integer, a - reference, l - long, f - float, d - double, c-char

?load\_x // lokale Variable a laden

?store\_x // Oberster Wert auf Stack in lokaler Variable a speichern

?const\_10 // Konstante 10 laden

bipush 10 // Konstante 10 laden

(iload, istore ... für integer)

## Arithmetische Berechnungen

?- steht für typ. i - integer, a - reference, l - long, f - float, d - double, c-char

a auf dem Stack, dann b auf dem Stack, dann sub

-> (a - b) auf dem Stack, a, b und sub - nicht mehr

Allgemein: old - new - op -> (old op new)

?add // Addition der zwei letzten Werte auf Stack

?sub // Subtraktion der zwei letzten Werte auf Stack

?mult // Multiplikation der zwei letzten Werte auf Stack

?div // Teilen der zwei letzten Werte auf Stack

?neg // Negieren Zahl

(iadd, imult, idiv, ineg... für integer)

## Sprungbefehle: 1 Parameter

x - das letzte Wert auf dem Stack

goto label //Sprung zu Sprungmarke label

ifeq label // x = 0 dann Sprung

ifge label // x >= 0 dann Sprung

ifgt label // x > 0 dann Sprung

ifle label // x <= 0 dann Sprung

iflt label // x < 0 dann Sprung

## Sprungbefehle: compare two parameters

zuerst wurde x geladen, dann wurde y geladen

i - integer, cmp - compare, eq - equal, g - greater, l - less, e - equal, t - than

if\_icmpeq label // x == y dann Sprung

if\_icmpge label // x >= y dann Sprung

if\_icmpgt label // x > y dann Sprung

if\_icmple label // x <= y dann Sprung

if\_icmplt label // x < y dann Sprung

## Sprungbefehle: nullwerte

ifnotnull label // Wenn nicht null dann sprung

ifnull label // Wenn null dann sprung

## Sprungbefehle: return

return // Return void

?return //Return Wert aus dem Stack

(ireturn - return integer)

## Methodenaufrufe

Objekt-Pointer soll auf dem Stack liegen, dann soll Parameter liegen, dann kommt invoke

Annahme: die Funktion foo() steht in Konstantenpool mit Nummer 12.

aload\_x // this. Pointer laden

invokevirtual #12 // Funktion an lokaler Konstanten Stelle x laden mit

invokestatic #12 // Statische Funk. an lokaler Konstanten Stelle x laden

invokespecial #12 // Konstruktor aufrufen

## Objekterzeugung

**WURDE IN VORLESUNG UND ÜBUNG NICHT BEHANDELT**

new #x // Neue Referenz von Typ x

dup // Oberster Stack Wert duplizieren wofür wird dup benutzt? -> Lerngruppe fragen

invokespecial #x // Konstruktor an Stelle x aufrufen

areturn // Referenz zurückgeben (a - reference)

## Sonstige Befehle

putfield FIELD:TYPE // Schreibe das Wert aus dem Stack in FIELD von TYPE

// wird oft mit this.FIELD gemeint

iinc v,n // inkrementiere die Variable mit der Adresse v um n (das neue Wert wird automatisch in v gespeichert)

--> v = v + n

iinc 3,1 // inkrementiere die Variable mit der Adresse 3 um 1

# FAQ

## Funktionen

### Wie kann man eine Funktion this.f aufrufen?

Annahmen:   
this-Pointer ist in Variable 0 gespeichert,  
Im Konstantenpool steht an Stelle 42 die Information für die nicht-stat Methode f.  
f braucht keine Argumente

aload\_0

invokevirtual #42

//dann liegt Ergebnis von f auf dem Stack

### Aufruf mit parameter?

Annahmen:

Signatur f(int arg)  
Argument liegt in Variable a mit Addresse 95

aload\_0

iload\_95

invokevirtual #42

//this laden -> parameter laden -> funktion

### Aufruf mit parameter, der inline berechnet wird?

Annahmen:

f(a \* (x - 1) + c)

a - 1, x - 2, c - 3

aload\_0

iload\_2

iconst\_1

isub

iload\_1

imult

iload\_3

iadd

invokevirtual #42

### Muss man ein „this“ hinzufügen, wenn es nicht explizit steht?

Ja, immer

## IFs und Sprungbefehle

### In welcher Rheihenfolge müssen Variablen auf Stack liegen?

„logische“ Rheihenfolge:

if (a > b) jump to x else y

a - 1, b - 2

iload 1

iload 2

if\_icmpgt x // integer compare greater than

goto y

### 

## Code generation

### Minimale vs nichtminimale Stackverbrauch

Neg (Add (Const 1) (Add (Const 2) (Var 3)))

-(1+(2+x))

Nichtminimal:

bipush 1 //

bipush 2 //

iload\_3 //

iadd

iadd

ineg

//es liegen 3 werte auf dem stack vor erstem iadd

Minimal:

bipush 2

iload\_3

iadd

bipash 1

iadd

ineg

//max 2 werte

# Code

## Objekt erstellen

**WURDE IN VORLESUNG UND ÜBUNG NICHT BEHANDELT**

Java:

**class** Test {

Test foo() {

**return new** Test();

}

}

Tabelle:

#1 java/lang/Object.<init>()V

#2 Test

#3 Test.<init>()V

Test() ausführen;

aload\_0

invokespecial #1;

return

Test foo();

new #2;

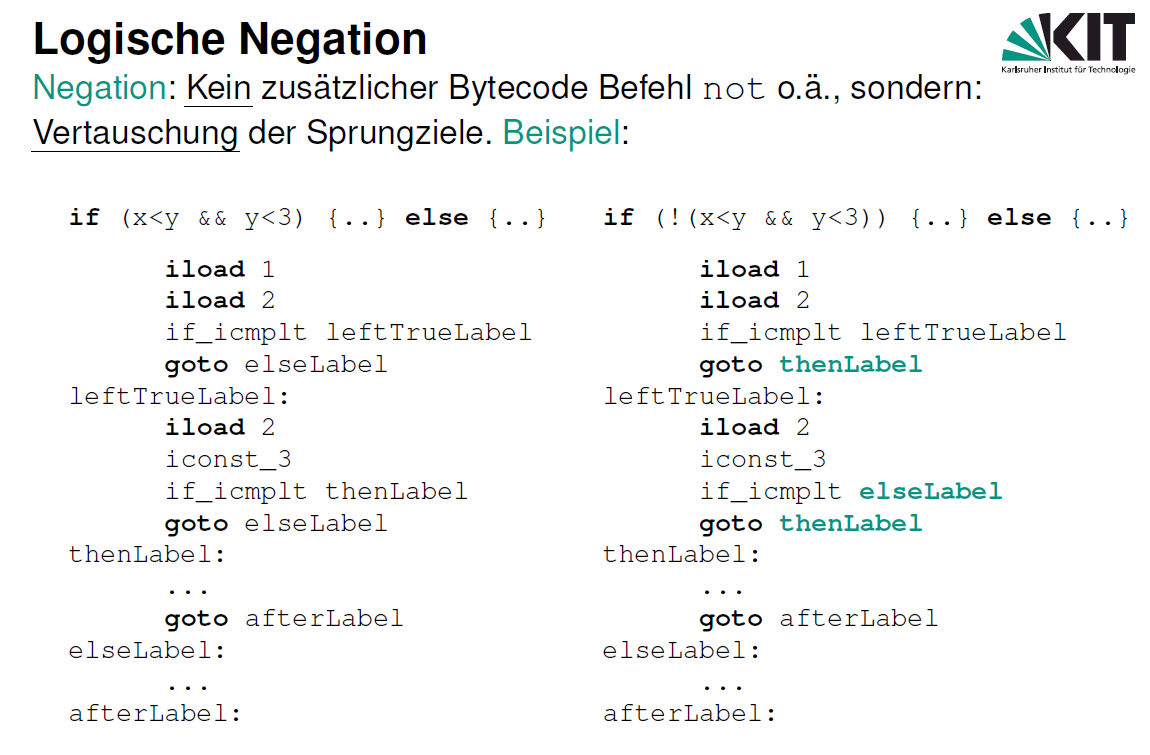
dup

invokespecial #3;

areturn

## Kurzauswertung und Negation

Immer von Links nach Rechts parsen und mehrmals die Labels überprüfen.



# Aufgaben

## || , && und not

**if** (((a < b) || !((a < c) || (c < b))) && !(c < 0)) {

c = b + a;

}

Hinweis: Um eine Bedingung der Form !*cond* zu übersetzen, genügt es, *cond* zu übersetzen und die Sprungziele zu tauschen.

Byte-Code:

