

# **toya: Eine imperative Programmiersprache für die JVM**

Lukas Christian Hofwimmer



## **MASTERARBEIT**

eingereicht am  
Fachhochschul-Bachelorstudiengang

Universal Computing

in Hagenberg

im Juli 2021

Betreuung:  
FH-Prof. DI Dr. Heinz Dobler

© Copyright 2021 Lukas Christian Hofwimmer

Diese Arbeit wird unter den Bedingungen der Creative Commons Lizenz *Attribution-NonCommercial-NoDerivatives 4.0 International* (CC BY-NC-ND 4.0) veröffentlicht – siehe <https://creativecommons.org/licenses/by-nc-nd/4.0/>.

# Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt. Die vorliegende, gedruckte Arbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Hagenberg, am 15. Juli 2021

Lukas Christian Hofwimmer

# Inhaltsverzeichnis

<b>Erklärung</b>	<b>iv</b>
<b>Vorwort</b>	<b>vii</b>
<b>Kurzfassung</b>	<b>viii</b>
<b>Abstract</b>	<b>ix</b>
<b>1 Einleitung</b>	<b>1</b>
<b>2 Die Programmiersprache toya</b>	<b>2</b>
2.1 Typen . . . . .	2
2.2 Funktionen . . . . .	3
2.3 Statements . . . . .	3
2.3.1 Variablen . . . . .	3
2.3.2 Arrays . . . . .	4
2.3.3 For-Schleifen . . . . .	5
2.4 Kommentare . . . . .	5
2.5 Ausdrücke . . . . .	5
2.5.1 Arithmetik . . . . .	5
2.5.2 Boolsche Logik . . . . .	5
2.5.3 If-Ausdrücke . . . . .	5
<b>3 Vergleich mit Kotlin</b>	<b>6</b>
<b>4 Generierung des Syntaxbaums mit ANTLR</b>	<b>7</b>
<b>5 Die JVM und deren Bytecode</b>	<b>8</b>
<b>6 Generierung des Bytecodes mit ASM</b>	<b>9</b>
<b>7 Implementierung von toya</b>	<b>10</b>
<b>8 Tests von toya</b>	<b>11</b>
<b>9 Fazit, Schlüsse und Lehren</b>	<b>12</b>
<b>A Technische Informationen</b>	<b>13</b>

Inhaltsverzeichnis	vi
<b>B Ergänzende Inhalte</b>	<b>14</b>
B.1 PDF-Dateien . . . . .	14
B.2 Mediendaten . . . . .	14
B.3 Online-Quellen (PDF-Kopien) . . . . .	14
<b>C Fragebogen</b>	<b>15</b>
<b>D LaTeX-Quellcode</b>	<b>16</b>
<b>Quellenverzeichnis</b>	<b>17</b>
Online-Quellen . . . . .	17

# Vorwort

# Kurzfassung



# Abstract

This should be a 1-page (maximum) summary of your work in English.

## Kapitel 1

### Einleitung

## Kapitel 2

# Die Programmiersprache toya

*Toya* ist eine stark typisierte, turing-vollständige Programmiersprache für die Java Virtual Machine mit einem Fokus auf Simplizität. Der Syntax ist, wie C# oder Java zum Beispiel, stark an C angelehnt. Auf den Syntax wird in diesem Kapitel bei der näheren Behandlung der einzelnen Komponenten der Sprache eingegangen.

Der Name *toya*, in Anlehnung an Java und Kotlin, findet seinen Ursprung bei einer Insel. Dabei handelt es sich konkret um 洞爺湖 (Tōya-ko), einen Kratersee im Norden Japans, der wiederum die Insel 中島 (Nakajima) beinhaltet. Von Tōya-ko leitet sich dann der Name *toya*.

Grundsätzlich folgt *toya* imperativen Programmierparadigmen. Ein *toya*-Programm besteht aus einer Menge an Funktionen und Variablen, wobei, wie in Java eine **main** Funktion zum Programmeinstieg benötigt wird. Funktionen beinhalten verschiedene Klassen gibt es keine; bei der Kompilation werden aber alle Programnteile in eine **Main** Klasse zusammengefasst, da jede *.class* Datei genau eine Klasse beinhalten muss. Sollte die **main** Funktion nicht vorhanden sein, so wird eine Exception während des Parsens geworfen. Variablen, die außerhalb von Funktionen definiert werden, können global verwendet werden. Global in diesem Kontext bedeutet, dass Variablen in allen Funktionen des Programms verwendet werden können.

### 2.1 Typen

*Toya* stellt insgesamt 5 Typen und deren Array-Gegenstücke zur Verfügung. Diese sind **boolean**, **int**, **char**, **double** und **string**, wobei hierbei **int**, **boolean** und **string** eindeutig die wichtigsten sind, da jeder dieser Typen in unterschiedlichen Domänen seine Verwendung findet. So werden **boolean** Das erstellen weiterer Typen ist nicht möglich.

Array-Typen werden über den allgemein bekannten Suffix ‘[]’ deklariert. So ist zum Beispiel der Typ eines String-Arrays als **string[]** zu schreiben. Die JVM bietet zusätzlich noch die Datentypen **byte**, **short**, **long** und **float** an, aber weil alle dieser Typen redundant in ihrem Verwendungszweck sind, kommen diese nicht in *toya* vor, da der Sinn von *toya* nicht die vollständige Ausschöpfung aller JVM-Features ist, sondern die explorative Implementierung einer Programmiersprache, wofür nicht alle Datentypen benötigt werden. Der **returnAddress** Typ wird hier außer acht gelassen, weil dieser nur JVM-interne Relevanz hat.

Integer besitzt einen Wertebereich von  $-2^{31}$  bis  $2^{31} - 1$ ; Double folgt der IEEE 754 Spezifikation: 1 Bit für das Vorzeichen 11 Bit für den Exponenten und 52 Bit für die Mantisse. Boolean kann die Werte `true` und `false` annehmen. `True` wird intern als 1 gehandhabt, `false` als 0. Die JVM besitzt keinen nativen String Typen, da dieser als Referenzwert gehandhabt wird. Da *toya* keine Erstellung von Typen erlaubt, sind die einzigen Referenztypen String und Arrays. Die Bytecode-Generierung *toya's* unterscheidet immer zwischen Arrays und nicht-Arrays, wenn es um die Auswahl der richtigen Opcodes geht, dadurch ergibt sich, dass eine Referenz, welche kein Array ist, immer ein String sein muss. Strings werden als Literale via doppelte Anführungszeichen definiert. So ist ein Hello World String als "Hello World" anzugeben.

## 2.2 Funktionen

Funktionen sind die zentrale Komponente von *toya* und beinhalten die Programmlogik. Sie bestehen aus einem Funktionskörper und der Funktionssignatur. Die Funktionssignatur beinhaltet Name der Funktion, Parameter und Rückgabewert. Der Name ist das einzig verpflichtende hierbei; Parameter und Rückgabewert sind rein optional. Hat eine Funktion keinen Rückgabewert, so ist in der Funktion der Pfeil, als auch der nachfolgende Typ wegzulassen.

**Listing 2.1:** Eine typische Funktion unter toya.

```
function add(lhs: int, rhs: int) -> int {  
    lhs + rhs  
}
```

Der Funktionskörper beinhaltet eine beliebige Menge an Statements und Ausdrücken. Hat eine Funktion einen Rückgabewert, so kann mit `return` ein nachfolgender Ausdruck rückgegeben werden. Das Schlüsselwort `return` ist jedoch optional: Wenn das letzte Statement gleichzeitig ein Ausdruck und vom geforderten Typen ist, dann wird automatisch dieser Ausdruck retourniert. Hierbei ist jedoch aufzupassen, dass die Leslichkeit erhalten bleibt.

Funktionen werden mit dem Syntax `<funcname>(parameter*)` aufgerufen und sind vom Rückgabetypen der aufgerufenen Funktion. Für jeden Parameter kann jeder beliebige Ausdruck eingesetzt werden, solange der Ist- und Solltyp übereinstimmt.

## 2.3 Statements

Statements sind Programmanweisungen, die keinen Wert zurückgeben. Dazu gehören Variablendeklaration und -zuweisung, For-Schleifen und Return-Anweisungen.

### 2.3.1 Variablen

*Toya* erlaubt der Nutzerin die Erstellung von Variablen in Funktionen und auf globaler Ebene. Der Syntax dafür lautet `var <name> = <ausdruck>`; die explizite Angabe eines Typens ist nicht möglich. Stattdessen inferiert der Compiler anhand bekannter Typinformation des zu evaluierenden Ausdrucks den Typen und weist diesen Typen der Variable zu. Dieser Typ bleibt über die gesamte Lebensdauer der Variable gleich. Sobald

der Typ einer Variable einmal fixiert ist, so kann dieser Typ nicht mehr geändert werden. Initialisiert man also eine Variable mithilfe eines Ausdrucks, der zu `int` evaluiert, so ist die Variable bis zur Vernichtung durch den Garbage Collector vom Typen `int`. Eine getrennte Deklaration und Initialisierung ist nicht möglich.

Abgesehen von typentheoretischer Relevanz bietet die Verwendung des Schlüsselwortes `var` einige Vorteile als auch Nachteile für Verwenderinnen von *toya*. Da `var` eine Vielzahl von verschiedenen Typen ersetzt, erleichtert es die Schreibarbeit für Programmiererinnen ungemein. Robert C. Martin sagt jedoch in seinem nominalen Werk *A handbook of agile software craftsmanship* “Code is more read than it is written.”. Daraus folgt, dass die Lesbarkeit wichtiger als *Schreibbarkeit* von Code ist und hierbei zeigen sich dann auch die Schwächen.

Ohne einer modernen Entwicklungsumgebung, welche via dem User-Interfaces Hinweise auf die Typisierung gibt, kann die Entwicklerin nur über den konkreten Typen Vermutungen anstellen. Aufgrund der geringen Anzahl an Typen in *toya* ist das Fehlen von Typhinweisen jedoch vernachlässigbar. Vergleicht man nun die Variablendeklaration und Initialisierung mit Java, so ist zu erkennen, dass bei der Initialisierung via Literalen die Verwendung von `var` kein Problem darstellt. Will man einer Variable den retournierten Wert einer Funktion zuweisen, so können hier aber Schwierigkeiten hinsichtlich Schlussfolgerungen auftreten. Daher ist die bewusste und intelligente Vergabe von Variablennamen essenziell.

**Listing 2.2:** Variablendeklaration in toya

```
var number = 123
var word = "Hello World"
var value = 123.456
var bool = true
var result = someFunction()
```

**Listing 2.3:** Variablendeklaration in Java (vor Version 10)

```
int number = 123;
String word = "Hello World";
double value = 123.456;
boolean bool = true;
int result = someFunction();
```

Der Name einer Variable darf nur einmal im eigenen Scope oder Elternscope verwendet werden, da es ansonsten zu Unklarheiten kommen kann, welche von mehreren Variablen nun gemeint ist. Auf die Frage, was sich hinter einem Scope verbirgt, wird näher im Kapitel 7 eingegangen. Ein Name darf aus beliebig vielen Groß- und Kleinbuchstaben und Unterstrichen bestehen.

### 2.3.2 Arrays

Arrays sind eine eindimensionale Liste von einem bestimmten Typen mit einer fixen Länge, welche bei der Deklaration des Arrays angegeben wird und sich über die Lebensdauer der Variable nicht mehr ändert. Zuweisungen und Deklarationen unterscheiden sich nur leicht von nicht-Array Variablen. Der Hauptunterschied dabei besteht darin, dass die Länge bei der Deklaration und der Index bei der Zuweisung angegeben werden muss.

Arrays werden mit dem Syntax `var <name> = new <typ>[int-Ausdruck]` deklariert. Mit `<name>[int-Ausdruck] = Ausdruck` wird ein neuer Wert auf einen Index geschrieben.

### 2.3.3 For-Schleifen

## 2.4 Kommentare

## 2.5 Ausdrücke

### 2.5.1 Arithmetik

### 2.5.2 Boolsche Logik

### 2.5.3 If-Ausdrücke

*Toya* erlaubt die Erstellung von Kommentaren. Ein Kommentar beginnt mit einem doppelten Schrägstrich `//`; alle weiteren Zeichen in dieser Zeile – bis zum Zeilenumbruch – ignoriert der Parser.

- introduction von toya (grundsätzliche features, „auf diese wird in diesem Kapitel noch näher eingegangen“) - namenserklärung - Funktionen - Typen - Variablen - Expressions: - Arithmetik - Boolean - If - Arrays - Statements - For Statement

## Kapitel 3

# Vergleich mit Kotlin

## Kapitel 4

# Generierung des Syntaxbaums mit ANTLR



## Kapitel 5

# Die JVM und deren Bytecode

## Kapitel 6

# Generierung des Bytecodes mit ASM

## Kapitel 7

# Implementierung von toya

Lorem ipsum dolor sit amet.

## Kapitel 8

### Tests von $t_{\alpha}$

## Kapitel 9

# Fazit, Schlüsse und Lehren

Anhang A

## Technische Informationen

## Anhang B

# Ergänzende Inhalte

Auflistung der ergänzenden Materialien zu dieser Arbeit, die zur digitalen Archivierung an der Hochschule eingereicht wurden (als ZIP-Datei).

### B.1 PDF-Dateien

Pfad: /

thesis.pdf . . . . . Finale Master-/Bachelorarbeit (Gesamtdokument)

### B.2 Mediendaten

Pfad: /media

\*.ai, \*.pdf . . . . . Adobe Illustrator-Dateien

\*.jpg, \*.png . . . . . Rasterbilder

\*.mp3 . . . . . Audio-Dateien

\*.mp4 . . . . . Video-Dateien

### B.3 Online-Quellen (PDF-Kopien)

Pfad: /online-sources

Reliquienschrein-Wikipedia.pdf

Anhang C

Fragebogen



Anhang D

LaTeX-Quellcode

# Quellenverzeichnis

## Online-Quellen

- [1] *Reliquienschrein*. 20. Okt. 2020. URL: <https://de.wikipedia.org/wiki/Reliquienschrein> (besucht am 12.05.2021).

# Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —