



UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Programmierungskurs Java

Methoden

Raphael Allner
Institut für Telematik
29. November 2019

1. Motivation

2. Blockanweisungen

- Gültigkeitsbereiche

3. Methoden

- Strukturierung und Wiederverwendung von Code
- Implementierung einer Methode
- Aufruf einer Methode
- Globale Variablen

Übungs- oder Hausaufgabe



UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

1. Laden Sie die Datei **Methoden.java** aus dem Moodle herunter
2. Verfolgen Sie die Vorlesung um die Aufgaben zu erfüllen
3. Überprüfen Sie Ihren Code
 1. Kompilieren mit `javac Methoden.java`
 2. Ausführen in der JVM mit `java Methoden`



Methoden

Motivation

Motivation



UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Bisher: Lange Abfolge von Code

Probleme:

- Lesbarkeit
- **Fehleranfälligkeit**
- Copy & Paste
- Keine zentrale Fehlerbehebung
- **Große Programme schwer zu warten**

Betriebssystem	Codezeilen
Windows NT 3.1	4-5 Mio.
Windows NT 4.0	11-12 Mio.
Windows 2000	>29 Mio.
Windows XP	45 Mio.
Windows Server 2003	50 Mio.
OpenSolaris	9.7 Mio.
FreeBSD	8.8 Mio.
Mac OS X 10.4	86 Mio.
Linux Kernel 2.6.0	5.2 Mio.
Linux Kernel 2.6.35	13.5 Mio.



Motivation

Beispiel: Trainingsstatistik



Vorab: Aufzeichnung der Laufzeiten über mehrere Trainingstage

- Tag 1: 16,5 14,3 15,0 14,5 13,9
- Tag 2: 16,5 20,3 17,0 23,6

Statistik:

- Beste Zeit
- Schlechteste Zeit
- Durchschnittszeit
- ...

Motivation

Beispiel: Quellcode Copy & Paste



```
1 public class LapTimes {
2     public static void main(String[] args) {
3
4         double[] lapTimes = {16.5, 14.3, 15.0, 14.5, 13.9};
5         double besteZeit = lapTimes[0];
6         for (int i = 1; i < lapTimes.length; i++) {
7             if (lapTimes[i] < besteZeit) {
8                 besteZeit = lapTimes[i];
9             }
10        }
11        System.out.println("Beste Rundenzeit: " + besteZeit);
12
13        double[] lapTimes2 = {16.5, 20.3, 17.0, 23.6};
14        besteZeit = lapTimes2;
15        for (int i = 1; i < lapTimes.length; i++) {
16            if (lapTimes2[i] < besteZeit) {
17                besteZeit = lapTimes2[i];
18            }
19        }
20        System.out.println("Beste Rundenzeit: " + besteZeit);
21
22
23    }
24 }
```



Motivation

Beispiel: Ausgabe des Programms



```
Terminal
File Edit View Search Terminal Help
$ gedit Laufzeiten.java
$ javac Laufzeiten.java
$ java Laufzeiten
Beste Rundenzeit: 13.9
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at Laufzeiten.main(Laufzeiten.java:16)
$
```

13,9 für Tag 1 ist OK

Zeit für den zweiten Tag wird nicht berechnet

- Zugriff auf Array-Element Nr. 5 (mit dem Index 4) nicht möglich
- Fehler in Zeile 16 in der Datei „LapTimes.java“

Wurde bei der Anpassung etwas vergessen?

Quellcode des Programms



```
1 public class LapTimes {
2     public static void main(String[] args) {
3
4         double[] lapTimes = {16.5, 14.3, 15.0, 14.5, 19.0};
5         double besteZeit = lapTimes[0];
6         for (int i = 1; i < lapTimes.length; i++) {
7             if (lapTimes[i] < besteZeit) {
8                 besteZeit = lapTimes[i];
9             }
10        }
11        System.out.println("Beste Rundenzeit: " + besteZeit);
12
13        double[] lapTimes2 = {16.5, 17.0, 23.6};
14        besteZeit = lapTimes2[0];
15        for (int i = 1; i < lapTimes2.length; i++) {
16            if (lapTimes2[i] < besteZeit) {
17                besteZeit = lapTimes2[i];
18            }
19        }
20        System.out.println("Beste Rundenzeit: " + besteZeit);
21
22    }
23 }
24 }
```

Copy & Paste-Fehler

Zeile 16



Methoden

Blockanweisungen und Gültigkeitsbereiche

Blockanweisungen definieren einen eigenen (Gültigkeits-) Bereich (engl. Scope)

Beispiele :

- **class** **MeinProgramm**
- Methode „**main**“

```
1 public class MeinProgramm {  
2     public static void main(String[] args){  
3         System.out.println("Hello World");  
4     }  
5 }  
6  
7 }
```

Blockanweisungen und Gültigkeitsbereiche

Blockanweisungen zur Strukturierung



```
1 public class LapTimeWithScope {
2     public static void main(String[] args) {
3         {
4             double[] lapTimes = {16.5, 14.3, 15.0, 14.5, 13.9};
5             double bestTime = lapTimes[0];
6             for (int i = 1; i < lapTimes.length; i++) {
7                 if (lapTimes[i] < bestTime) {
8                     bestTime = lapTimes[i];
9                 }
10            }
11            System.out.println("Beste Rundenzeit: " + bestTime);
12        }
13        {
14            double[] lapTimes = {16.5, 20.3, 17.0, 23.6};
15            double bestTime = lapTimes[0];
16            for (int i = 1; i < lapTimes.length; i++) {
17                if (lapTimes[i] < bestTime) {
18                    bestTime = lapTimes[i];
19                }
20            }
21            System.out.println("Beste Rundenzeit: " + bestTime);
22        }
23    }
24 }
```

Die Farben der geschweiften Klammern kennzeichnen Anfang und Ende einzelner Gültigkeitsbereiche

Lokale Variablen, die in einem Gültigkeitsbereich **deklariert** werden, sind außerhalb nicht gültig

In diesem Fall Blockanweisungen **nur bedingt geeignet**

- Keine Wiederverwendung von Code
- Code-Verdoppelung bleibt
- Verschachtelung von Gültigkeitsbereichen schwer zu überblicken

Besser:

- Unterteilung in wiederverwendbare, funktionale Einheiten



Methoden

Methoden

Methoden

Strukturierung und Wiederverwendung

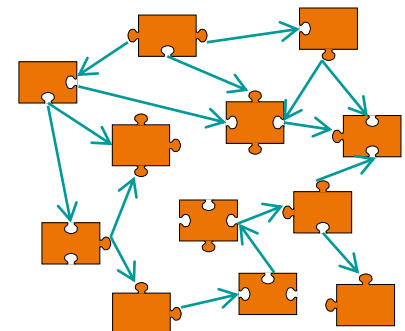


UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Zerlegung von Programmen in kleinere, wiederverwendbare Einheiten

Idee:

- **Kapselung** einzelner Funktionalitäten
- Bessere **Wiederverwendbarkeit**
- Lesbarer Code
- Fehlerbehebung an einer Stelle



Methoden

Strukturierung und Wiederverwendung



输入：“变量”带着确定的数据类型

Eingabewerte (auch Eingabeparameter)

- Übergabe von **Werten** an die Methode
- Definiert als **Menge von Variablen** mit **bestimmtem Datentyp**

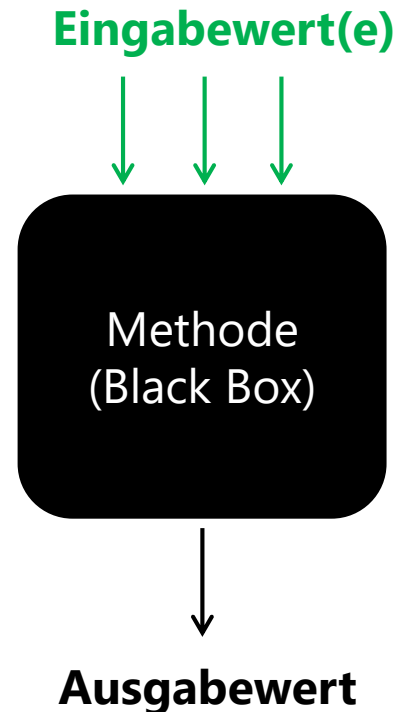
输出：一个值和确定的数据类型

Ausgabewert (auch Ausgabeparameter / Rückgabe)

- **Wert** mit einem **bestimmten Datentyp**
- Spezieller Datentyp für keine Rückgabe: **void**
void: 非返回值

Beispiele für Deklarationen:

```
void main(String[] args);  
int countSpaces(char[] text);  
char[] reverse(char[] text);  
boolean isASpace(char character);  
boolean getBalance(int account, int bankCode);
```



Methoden

Implementierung einer Methode



Methoden sind wie eine „Erweiterung“ der Blockanweisung

- Funktionale Einheit
- Kann von anderen Programmteilen genutzt werden
- Parameter werden wie normale lokale Variablen verwendet
 - Werte der Parameter werden beim Aufruf festgelegt

Beispiel:

```
10  double getBestLapTime(double[] 输入变量 lapTimes){  
11      double bestTime = lapTimes[0];  
12  
13      for (int i = 1; i < lapTimes.length; i++) {  
14          if (lapTimes[i] < bestTime) {  
15              bestTime = lapTimes[i]; 运行过程  
16          }  
17      }  输出值  
18      return bestTime;  
19  }
```

Methoden

Methodenkopf und -rumpf



Kopf

```
10 double getBestLapTime(double[] laufzeiten){  
11     // Hier steht die Implementierung der Methode  
12     // in einer Blockanweisung  
13 }
```

Rumpf
(auch: Implementierung)

Methoden

Beispiel



```
10 double getBestLapTime(double[] lapTimes){
11     double besteZeit = lapTimes[0];
12
13     for (int i = 1; i < lapTimes.length; i++){
14         if (lapTimes[i] < besteZeit){
15             besteZeit = lapTimes[i];
16         }
17     }
18     return besteZeit;
19 }
```

Datentyp entsprechend der
Methode

Strukturierung und Wiederverwendung

Aufruf von Methoden



```
1 public class LapTimeWithMethod {
2
3     public static double getBestLapTime(double[] times){
4         double bestTime = times[0];
5         for (int i = 1; i < times.length; i++) {
6             if (times[i] < bestTime) {
7                 bestTime = times[i];
8             }
9         }
10        return bestTime;
11    }
12
13    public static void main(String[] args) {
14
15        double[] lapTimes = {16.5, 24.3, 15.0, 14.5, 13.9};
16        double bestTime = getBestLapTime(lapTimes);
17        System.out.println("Beste Rundenzeit: " + bestTime);
18
19        double[] lapTimes2 = {16.5, 20.3, 17.0, 23.6};
20        System.out.println("Beste Rundenzeit: " + getBestLapTime(lapTimes2));
21    }
22 }
```

运行 Methode
Methodenaufruf
über den Namen (Identifier)

Übergabe von **Parametern**
an die **Methode**.

Zu beachten:

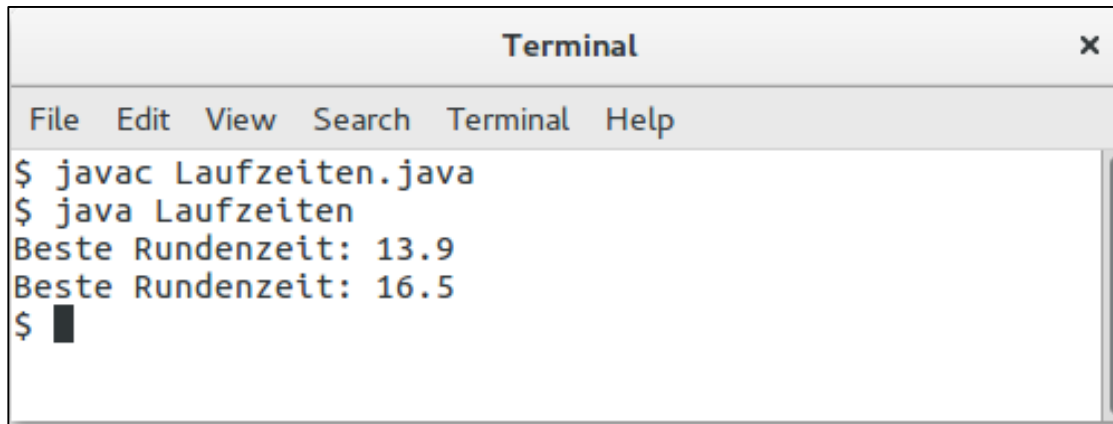
- Richtige Anzahl
- Richtiger **Datentyp**

正确的值
正确的数据类型

Programm verhält sich identisch zum Copy & Paste-Programm

Aber das Programm ist ...

- ... **lesbarer**
- ... **wiederverwendbarer** und
- ... einfacher **wartbarer** bzw. **weiterzuentwickeln**,



```
Terminal
File Edit View Search Terminal Help
$ javac Laufzeiten.java
$ java Laufzeiten
Beste Rundenzeit: 13.9
Beste Rundenzeit: 16.5
$
```

Strukturierung und Wiederverwendung

Kommunikation über globale Variablen



```
1 public class LapTimesWithGlobalVariables {
2
3     public static double totalBest = 3000;
4
5     public static double getBestLapTime(double[] times){
6         double bestTime = times[0];
7         for (int i = 1; i < times.length; i++) {
8             if (times[i] < bestTime) {
9                 bestTime = times[i];
10            }
11        }
12        if (bestTime < totalBest) {
13            totalBest = bestTime;
14        }
15        return bestTime;
16    }
17
18    public static void main(String[] args) {
19        double[] lapTimes = {16.5, 14.3, 15.0, 14.5, 13.9};
20        double bestTime = getBestLapTime(lapTimes);
21        System.out.println("Beste Rundenzeit: " + bestTime);
22
23        double[] lapTimes2 = {16.5, 20.3, 17.0, 23.6};
24        System.out.println("Beste Rundenzeit: " + getBestLapTime(lapTimes2));
25        System.out.println("Beste Rundenzeit insgesamt: " + totalBest);
26    }
27 }
```

Auf die **globalen Variablen** kann innerhalb der Klasse immer zugegriffen werden

Strukturierung und Wiederverwendung

Beispielprogramm



```
1 public class GreetAndAdd {
2
3     public static int totalSum = 0;
4
5     public static void main(String[] args) {
6         String n = args[0];
7         greet(n);
8         greet(args[0]);
9         int nr1 = 1;
10
11         int sum1 = add(nr1, 2);
12         {
13             int sum2 = add(3, 4);
14         }
15         // hier besteht kein Zugriff mehr auf 'sum2'
16     }
17
18     public static void greet(String name){
19         System.out.println("Hallo " + name);
20     }
21
22     public static int add(int z1, int z2) {
23         int sum = z1 + z2;
24         totalSum += sum;
25         return totalSum;
26     }
27 }
```

Bisher:

- Aufteilung des Programms in **Methoden** mit bestimmten Aufgaben
- Hauptprogramm steuert **Programmfluss** über **Methodenaufrufe**
- **Wiederverwendung** implementierter Funktionalität
- Übergabe von Daten über **Parameter** oder **globale Variablen**

Probleme:

- Verwendung globaler Variablen ist problematisch
 - Unklar, **wo** sie verändert werden
 - Änderungen wirken sich ggf. auf andere Methoden aus
- Bisher **alle Methoden in einer Klasse**
 - Strukturierung für sehr umfangreiche Programme reicht nicht aus



UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

Kontakt

Raphael Allner, M. Sc.
Wissenschaftlicher Mitarbeiter
Institut für Telematik

Universität zu Lübeck
Ratzeburger Allee 160
23562 Lübeck

<https://www.itm.uni-luebeck.de/mitarbeitende/raphael-allner.html>

