hhu.

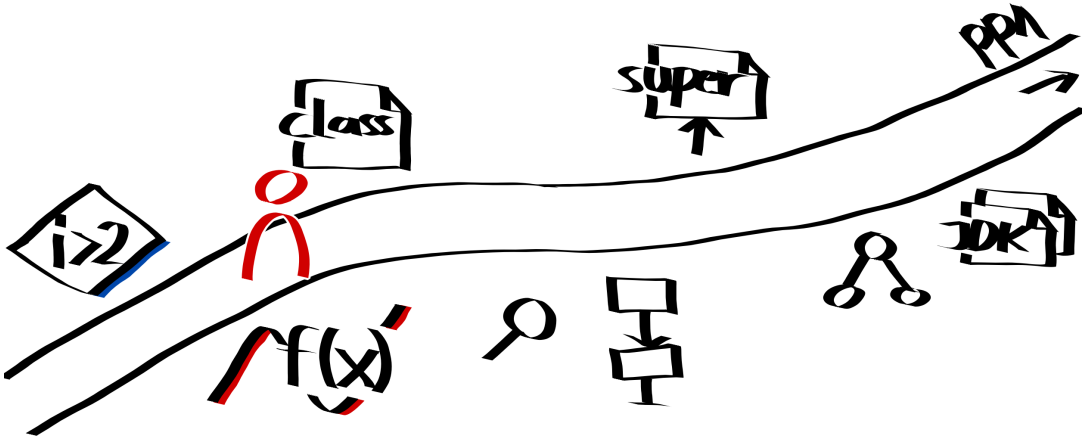
Kapitel 2: Methoden & IO

VL 8: Überladen von Methoden

Wintersemester 2023/24

Stand 6. November 2023, 07:01 Uhr

Wo stehen wir gerade?



Vorlesungsfeedback

Heinrich Heine
Universität Düsseldorf

- Tempo/Schwierigkeit: leichte Tendenz zu hoch (n=6)
- Lotto: Nächstes Jahr wird „verschiedene“ fett gedruckt.

\+ Zusatzmaterial

\+ spaßige, lockere VL, Wiederholung

– Schwierigkeit/Zeitaufwand der Aufgaben

? längere Beispiele mit mehr Kommentaren hochladen

? Syntax-Hilfsblatt

? `InputMismatchException` beim Sum-Beispiel

i Korrekturen brauchen noch bis morgen

i Dachdecker

Überladen von Methoden[1]

Heinrich Heine Universität Düsseldorf

Definition

Eine Methode wird **überladen**, wenn es zwei Methoden mit demselben Namen, aber unterschiedlichen Parameterlisten (Typ oder Anzahl) gibt. D. h. es kann Methoden mit demselben Namen, aber unterschiedlichen Signaturen mehrfach pro Datei geben.

Anmerkungen:

- Pro Klasse Methodensignaturen einmalig
- Rückgabetyp kann nicht überladen werden

[1] Overloading (ad-hoc Polymorphismus)

Beispiel: Summe I

Overloading.java — Java

```java
public class Overloading {
    private static int sum(int a, int b) {
        System.out.println("Signatur: sum(int, int)");
        return a + b;
    }

    private static int sum(int a, int b, int c) {
        System.out.println("Signatur: sum(int, int, int)");
        return a + b + c;
    }

    private static double sum(double a, double b) {
        System.out.println("Signatur: sum(double, double)");
        return a + b;
    }

    public static void main(String[] args) {
        System.out.println(sum(1, 2));
        System.out.println(sum(1, 2, 3));
        System.out.println(sum(1, 2.2)); // automatischer Cast von 1 zu Double
    }
}
```

Beispiel: Summe II

Heinrich Heine
Universität Düsseldorf

```
% java Overloading
Signatur: sum(int, int)
3
Signatur: sum(int, int, int)
6
Signatur: sum(double, double)
3.2
```

Eigentlich schon die ganze Zeit benutzt . . .

- `Math.abs`:
  - `Math.abs(-2)` verwendet `int abs(int)`
  - `Math.abs(-2.1)` verwendet `double abs(double)`
- `System.out.println`:
  - `System.out.println(1)` verwendet `System.out.println(int)`
  - `System.out.println(1.2)` verwendet `System.out.println(double)`
  - `System.out.println(true)` verwendet `System.out.println(boolean)`
- Java wählt richtige Methode automatisch basierend auf Typ + Anzahl der Argumente

Gibt es hier ein Problem?

```
2  private static int sum(int a, int b) {
3      System.out.println("Signatur: sum(int, int)");
4      return a + b;
5  }
6
7  private static double sum(int a, int b) {
8      System.out.println("Signatur: sum(int, int)");
9      return a + b;
10 }
11
12 public static void main(String[] args) {
13     System.out.println(sum(1, 2));
14 }
```

Gibt es hier ein Problem?

Heinrich Heine Universität Düsseldorf

```java
private static int sum(int a, int b) {
    System.out.println("Signatur: sum(int, int)");
    return a + b;
}

private static double sum(int a, int b) {
    System.out.println("Signatur: sum(int, int)");
    return a + b;
}

public static void main(String[] args) {
    System.out.println(sum(1, 2));
}
```

```
NoOverloading1.java:7: error: method sum(int,int) is already defined in class
  NoOverloading1
    private static double sum(int a, int b) {
                          ^
```

Und hier?

```
2  private static double sum(int a, int b) {
3      System.out.println("Signatur: sum(int, int)");
4      return a + b;
5  }
6
7  private static double sum(int number1, int number2) {
8      System.out.println("Signatur: sum(int, int)");
9      return number1 + number2;
10 }
11
12 public static void main(String[] args) {
13     System.out.println(sum(1, 2));
14 }
```

Und hier?

```java
private static double sum(int a, int b) {
    System.out.println("Signatur: sum(int, int)");
    return a + b;
}

private static double sum(int number1, int number2) {
    System.out.println("Signatur: sum(int, int)");
    return number1 + number2;
}

public static void main(String[] args) {
    System.out.println(sum(1, 2));
}
```

```
NoOverloading2.java:7: error: method sum(int,int) is already defined in class
  NoOverloading2
    private static double sum(int number1, int number2) {
                          ^
```

Beispiel: Mehrdeutigkeiten I

```java
public class ConfusingOverloading {
    private static double sum(int a, double b) {
        System.out.println("Signatur: sum(int, double)");
        return a + b;
    }

    private static double sum(double a, int b) {
        System.out.println("Signatur: sum(double, int)");
        return a + b;
    }

    public static void main(String[] args) {
        System.out.println(sum(1, 2)); // Compilezeitfehler
        System.out.println(sum((double)1, 2)); // funktioniert
    }
}
```

Beispiel: Mehrdeutigkeiten II

```
% javac ConfusingOverloading.java
ConfusingOverloading.java:14: error: reference to sum is ambiguous
        System.out.println(sum(1, 2));
                           ^
  both method sum(int,double) in ConfusingOverloading and method sum(double,int)
    in ConfusingOverloading match
1 error
```

Verhalten genau definiert[2]

[2] https://docs.oracle.com/javase/specs/jls/se8/html/jls-15.html#jls-15.12.2

Zerschlagung in Teilprobleme

Heinrich Heine
Universität Düsseldorf

Aufgabe

Speichere alle Primzahlen kleiner 100 in einem Array.

Zerschlagung in Teilprobleme

Heinrich Heine
Universität Düsseldorf

Aufgabe

Speichere alle Primzahlen kleiner 100 in einem Array.

1. Bestimme Anzahl *n* der Primzahlen kleiner 100

Zerschlagung in Teilprobleme

Heinrich Heine
Universität Düsseldorf

Aufgabe

Speichere alle Primzahlen kleiner 100 in einem Array.

1. Bestimme Anzahl *n* der Primzahlen kleiner 100
   - Prüfe für alle Zahlen kleiner 100, ob prim

Zerschlagung in Teilprobleme

Heinrich Heine Universität Düsseldorf

Aufgabe

Speichere alle Primzahlen kleiner 100 in einem Array.

1. Bestimme Anzahl *n* der Primzahlen kleiner 100
   - Prüfe für alle Zahlen kleiner 100, ob prim
     - Prüfe, ob eine bestimmte Zahl prim ist

Zerschlagung in Teilprobleme

Heinrich Heine
Universität Düsseldorf

Aufgabe

Speichere alle Primzahlen kleiner 100 in einem Array.

1. Bestimme Anzahl *n* der Primzahlen kleiner 100
   - Prüfe für alle Zahlen kleiner 100, ob prim
     - Prüfe, ob eine bestimmte Zahl prim ist
       - Bestimme die Anzahl der natürlichen Teiler einer Zahl

Zerschlagung in Teilprobleme

Heinrich Heine
Universität Düsseldorf

Aufgabe

Speichere alle Primzahlen kleiner 100 in einem Array.

1. Bestimme Anzahl *n* der Primzahlen kleiner 100
   - Prüfe für alle Zahlen kleiner 100, ob prim
     - Prüfe, ob eine bestimmte Zahl prim ist
       - Bestimme die Anzahl der natürlichen Teiler einer Zahl
2. Berechne Array für die ersten *n* Primzahlen

Zerschlagung in Teilprobleme

Heinrich Heine
Universität Düsseldorf

Aufgabe

Speichere alle Primzahlen kleiner 100 in einem Array.

1. Bestimme Anzahl *n* der Primzahlen kleiner 100
   - Prüfe für alle Zahlen kleiner 100, ob prim
     - Prüfe, ob eine bestimmte Zahl prim ist
       - Bestimme die Anzahl der natürlichen Teiler einer Zahl
2. Berechne Array für die ersten *n* Primzahlen
   - Suche so lange Primzahlen, bis *n* Stück gefunden

Zerschlagung in Teilprobleme

hhu
Heinrich Heine
Universität Düsseldorf

Aufgabe

Speichere alle Primzahlen kleiner 100 in einem Array.

1. Bestimme Anzahl *n* der Primzahlen kleiner 100
   - Prüfe für alle Zahlen kleiner 100, ob prim
     - Prüfe, ob eine bestimmte Zahl prim ist
       - Bestimme die Anzahl der natürlichen Teiler einer Zahl
2. Berechne Array für die ersten *n* Primzahlen
   - Suche so lange Primzahlen, bis *n* Stück gefunden
     - Prüfe, ob eine bestimmte Zahl prim ist (s. o.)

Zerschlagung in Teilprobleme

Heinrich Heine
Universität Düsseldorf

Aufgabe

Speichere alle Primzahlen kleiner 100 in einem Array.

1. Bestimme Anzahl *n* der Primzahlen kleiner 100
   - Prüfe für alle Zahlen kleiner 100, ob prim
     - Prüfe, ob eine bestimmte Zahl prim ist
       - Bestimme die Anzahl der natürlichen Teiler einer Zahl

2. Berechne Array für die ersten *n* Primzahlen
   - Suche so lange Primzahlen, bis *n* Stück gefunden
     - Prüfe, ob eine bestimmte Zahl prim ist (s. o.)
       - Bestimme die Anzahl der natürlichen Teiler einer Zahl (s. o.)

Methoden & Teilprobleme

Heinrich Heine
Universität Düsseldorf

Sinnvoll: Jedes Teilproblem in einer Methode lösen.

- einzelne Methoden übersichtlicher
- 1 Methode = 1 (kleine) Aufgabe
- Wiederverwendbarkeit von Code
- einfachere (automatische) Testbarkeit ($\rightarrow$ Programmierpraktikum)

Debugging

Heinrich Heine
Universität Düsseldorf

- Bug: Fehler in einem Computerprogramm
- Debugging: Systematisches finden und Entfernen von Laufzeitfehlern
  - Teilfunktionalitäten (Methoden) auf Richtigkeit testen
    - am besten immer zwischendurch; nicht erst, wenn komplettes Programm fertig
  - Prüfen, ob Variablen erwartete Werte haben
    - fürs Erste: Ausgabe von Variablenwerten mit `System.out.println`
    - später: Debugger der IDE

Fremden Code lesen

- beginne bei der main-Methode
- Methode von oben nach unten lesen
  - Variablen- und Methodennamen verraten, was passiert
  - sollte sich wie ein Text lesen können
  - nur wenn man sich für Details interessiert: Implementierung von aufgerufenen Methoden ansehen

$\Rightarrow$ Wenn Ihre Großeltern (die Englisch können) Ihre Methoden verstehen, sind sie gut lesbar.

Lernziele

Heinrich Heine
Universität Düsseldorf

Sie können am Ende der Woche ...

- das Überladen von Methoden **erklären**.
- Probleme der realen Welt durch Zerschlagung in Teilprobleme programmatisch **lösen**.
- Fehler in Quellcode **beheben**.
- die Funktionalität von fremden Quellcode **erklären**.

Wichtige Begriffe

überladen Zerschlagung in Teilprobleme Bug

Debugging

Übungsblatt

- Segelboot
- Histogramm
- Kniffel