Einführung in Java

Arne Hüffmeier

Michelle Liebers, Dennis Hoffmann

Tilman Lüttje, Jean Wiele

Angelehnt an Java-Vorkurs der Freitagsrunde

Wie war das noch gleich?

Nun können wir Programme schreiben, aber was bringt uns all das, wenn wir später nicht mehr wissen, was wir getan haben?

Man bräuchte eine Art Notizzettel, um sich merken zu können, was im Quelltext steht.

Die Lösung: Kommentare!

Eine alte Methode

Gucken wir uns ein Beispiel an:

```
public static int f(int x) {
    int v1, v2, v3;
    v1 = 1;
    v2 = 1;
    for (int i = 2; i < x; i++) {
        v3 = v1;
        v1 += v2;
        v2 = v3;
    }
    return v1;
}
```

Wer weiß was diese Methode ausgibt?

Ist es gut zu erkennen?

Die Kommentare

Ein paar Kommentaren machen diese Methode deutlich übersichtlicher

Inlinekommentar

```
// Kommentar bis zum Ende der Zeile
```

- Oft mitten im Code zu finden
- Meistens ein Hinweis was an der Stelle passiert

Blockkommentar

```
/* Geht ueber
   mehrere Zeilen */
```

- Ein mehrzeilige Beschreibung der Vorgänge an der Stelle.
- Kann auch innerhalb einer Zeile stehen, da er nur vom /* bis zum */ geht.

JavaDoc

Es gibt noch einen dritten Kommentartyp, den **JavaDoc**-Kommentar.

```
/**
 * Ein Satz der die Klasse oder Methode beschreibt.
 * Viele weitere zum genauen Erklaeren.
 *
 * @author Arne Hueffmeier
 *
 */
```

- JavaDoc wird verwendet, um anderen Hilfestellung zu geben, beim Umgang mit dem eigenen Programm, oder sich selbst.
- JavaDoc unterscheidet sich von normalen Blockkommentaren, dass er mit /** statt mit /* eingeleitet wird.
- JavaDoc schreibt man immer über die zugehörige Methode oder Klasse.

Übersicht über JavaDoc Besonderheiten

@author	Angabe über den Autor der Klasse.
@param x	Angabe über den Parameter x, was der sein soll, oder was damit geschehen wird.
@return	Angabe über den Rückgabewert der Methode.
@version	Angabe zur Version des Programms.

Zudem hat ein Satz in JavaDoc mit einem Punkt beendet zu werden.

Gucken wir uns den Code noch einmal an.

Eine alte Methode

```java
/**
 * Methode zum Ausgeben der Fibonaccizahl von x.
 *
 * @param x Zahl von der wir die Fibonaccizahl wollen.
 * @return Die Fibonaccizahl an stelle x.
 */
public static int f(int x) {
    //Variablen zum arbeiten.
    int v1, v2, v3;
    v1 = 1; // initialisieren der Variablen mit 1.
    v2 = 1; // fib(1) = 1 && fib(2) = 1
    /* Solange wir noch nicht die x. zahl erreicht haben.
       und x groesser als 2 ist */
    for (int i = 2; i < x; i++) {
        // v1 ist die groessere Zahl.
        v3 = v1; //v1 zwischenspeichern
        v1 += /* v1 + */ v2; //v2 auf v1 aufaddieren
        v2 = v3; //das alte v1 in v2 abspeichern
        //naechster Iterationsschritt
    }
    //die groessere der beiden Fibonaccizahlen ist in v1
    //und damit die an Stelle x.
    return v1;
}
```

Eine alte Methode

Deutlich übersichtlicher so, oder?

Tut euch selber und allen, die euren Code irgendwann mal lesen werden, den Gefallen und kommentiert ihn ausführlich.

Die richtige Ausrichtung

Hier ist ein Fehler enthalten:

```
public static int f(int x) {
int k = 0;
for (int i = 0; i <= x; i++) {
if (i % 2 == 0) {
k = k + i;
}
}
```

Dieser Fehler wäre offensichtlicher gewesen, hätte man eingerückt.

```
public static int f(int x) {
    int k = 0;
    for (int i = 0; i <= x; i++) {
        if (i % 2 == 0) {
            k = k + i;
        }
    }
```

Wir sehen sofort, da fehlt eine Klammer!

NetBeans

NetBeans ist nett zu uns. Es hat extra eine Funktion zur Autoformatierung, welche über das Tastenkürzel

Alt + Shift + F

den Quellcode richtig formatiert.

Richtig benennen

```
String Meine-Oma-faehrt-im-Huehnerstall-Motorrad = "Hallo Welt";
```

Eine sinnvolle Benennung von Variablen, Methoden oder Klassen gehört zum guten Ton.
Auch Java hat einige Konventionen, die, wenn man sich an sie hält, lesbaren und verständlichen Code produzieren.

Richtig benennen

- Variablen und Methodennamen mit Kleinbuchstaben anfangen.
- Variablen sinnvoll nach ihrem Verwendungszweck benennen.
- Methoden ebenso.
- Klassen großschreiben.
- Zählvariablen i,j,k nennen.
- CamelCase statt Unter- oder Bindestriche verwenden.

CamelCase

Was ist dieses CamelCase eigentlich?

CamelCase ist eine Schreibweise von zusammengesetzten Wörtern.

So wird im CamalCase aus

gib mir Fisch

durch das Streichen der Leerzeichen und Großschreiben des ersten Buchstaben des nächsten Wortes

gibMirFisch

Beispiele für gute Namen

GUT	
Fehler ArrayIndexOutOfBoundsException	Index eines Arrays ist außerhalb der Grenze
Methode isCharInWord	Ist das Zeichen im Wort?
Variable isInWord	Impliziert Ja/Nein-Frage → boolean

schlecht	
Methode convert	Konvertiert zu was?
Variable foo	Könnte alles mögliche sein

Ein besseres Array

Java gibt uns eine Möglichkeit Arrays mit variabler Länge zu erstellen.

Die ArrayList ist einem Array sehr ähnlich.

Wir können

- die Länge ändern.
- auf einzelne Elemente zugreifen.
- sie automatisch sortieren lassen.

Wie bekomme ich diese Liste?

Nun erklären wir euch, wie man eine ArrayList nutzt.
Zur Vorbereitung:

```
import java.util.ArrayList;
```

Erklärung

import : sagt Java, dass wir etwas von außerhalb unserer Klasse verwenden wollen.

java.util.ArrayList : ist der Verweis auf die Quelle, die wir verwenden wollen.

Diese Anweisung kommt über den Beginn der Klasse.

In Code

Hier ein Beispiel:

```
import java.util.ArrayList;
public class Test {
    private ArrayList<Integer> arrayList;

    public Test() {
        arrayList = new ArrayList<>();
    }
}
```

ArrayList erklärt

ArrayList<Integer> : Dies deutet an, dass wir eine ArrayList mit Integern machen wollen.
Ähnlich wie **int[]**.

ArrayList<>() : Konstruktor der ArrayList. Gleichbedeutend wie ArrayList<Integer>();
Dies erstellt uns eine Liste der Länge 0.

Der Harken

Warum schreibe ich da Integer und nicht einfach int?

> *Weil eine ArrayList nur mit Objekten funktioniert. Ein int ist kein Objekt, darum nutzen wir Integer, welcher eine Objektvariante des int ist.*

Und was bringt mir das für Vorteile?

> *Ein Array mit einer beliebigen Länge!*

So geht es

Hinzufüegen

Einige nützliche Beispiele:

Eintrag hinzufügen

```
arrayList.add(5);
```

5 wird ans Ende des Arrays hinzugefügt.

Eintrag abfragen

```
arrayList.get(0);
```

Gibt das erste Element der Liste zurück.

Hinzufüegen

Einige nützliche Beispiele:

Länge abfragen

```
arrayList.size();
```

Gibt uns die Anzahl der Einträge in unserer ArrayList zurück.

Eintrag entfernen

```
arrayList.remove(0);
```

Löscht das erste Element der ArrayList.

Weitere Features

Kann die auch mehr?

Ja, in der Java-API steht noch viel mehr.

Die Bahn ist nicht zufrieden

Es ist zwar schön, dass wir die Wagen eine bestimmte Klasse zuteilen können, aber die Deutsche Bahn will nun Doppelstockwaggons verwenden.

Wie setzen wir das nun um?

Wir wollen nur ungern wieder von vorn anfangen, also verwenden wir unseren Waggon weiter!
Betrachten wir die Unterschiede

- Wir können oben und unten unterschiedliche Klassen haben
- Wir können oben und unten unterschiedlich viele Passagiere haben

Eine Idee:

Wir erweitern den Waggon um passagiereOben und klasseOben!

Der neuere Waggon

So sähe unser Waggon wie folgt aus

```
public class Waggon {
private int klasse, klasseOben;
private int passagiere, passagiereOben;

public Waggon(int klasse, int klasseOben) {
this.klasse = klasse;
this.klasseOben = klasseOben;
passagiere = 0;
passagiereOben = 0;
}
}
```

Natürlich werden auch die Setter und Getter Methoden erweitert um diese beiden Attribute.

Ein schwieriger Kunde

Kaum ist unsere Version des Doppelstockwaggons fertig, ruft der Bahnchef wutentbrannt bei uns an.

Er beschwert sich, dass er nun nur noch Doppelstockwaggons verwenden kann, aber ein ICE keine hat. Wir sollen dieses Problem umgehend beheben, aber die Möglichkeit lassen, einen Zug mit Doppelstockwaggons ausstatten zu können.

Wir bräuchten also weiterhin unseren alten Waggon und müssen einen Weg finden, einen Sonderfall davon zu erstellen, den wir Doppelstockwaggon nennen sollen.

Java gibt uns da eine Möglichkeit. Sie nennt sich Vererbung.

Vererbung

Wir haben schnellstmöglich unseren Waggon wieder zurück gebaut. Nun kommt der zweite Versuch, einen Doppelstockwaggon zu erstellen.

Zweiter Versuch eines Doppelstockwaggons

```
public class Doppelstockwaggon extends Waggon {
private int klasseOben;
private int passagiereOben;

public Doppelstockwaggon(int klasse, int klasseOben) {
super(klasse);
this.klasseOben = klasseOben;
passagiereOben = 0;
}
```

Doppelstockwaggon erklärt

extends sagt Java, dass unser Doppelstockwaggon von Waggon erbt, also alles kann, was Waggon kann und auch als Waggon behandelt werden kann.

super() ruft den Konstruktor der Klasse auf, von der wir geerbt haben. In dem Fall wird also dadurch ein Waggon erstellt, welchen wir nur ergänzen.

Die Klasse Waggon wird aus Sicht des Doppelstockwaggons als Superklasse bezeichnet, weil wir den Konstruktor des Waggons mit super aufrufen.

Die Java API

Das beste Nachschlagewerk für Java.

http://docs.oracle.com/javase/8/docs/api/

oder nach **Java API** googlen.
'Java ist eine Insel' ist ebenfalls sehr empfehlenswert

http://openbook.galileocomputing.de/javainsel/