Javaschubla.de - Java als erste Programmiersprache

Statische Methoden

Methoden sind Funktionen innerhalb von Klassen, und weil bei Java alle Funktionen in Klassen stehen und nie außerhalb, gibt es bei Java nur Methoden. Ein anderer Begriff für Methode ist Elementfunktion.

Andere Begriffe für Funktion sind Prozedur, Subroutine und Unterprogramm. Je nach Art der Programmiersprache werden andere Begriffe verwendet, es gibt auch kleine Unterschiede, die hier aber egal sind.

In diesem Artikel soll es erstmal nur um statische Methoden gehen, nicht-statische Methoden kommen als nächstes OO-Thema. "static" wird weiter unten ansatzweise erklärt, aber wahrscheinlich wird es erst später klar.

Eine statische Methode hatten wir schon in allen Beispielprogrammen (außer beim Applet): Die main-Methode. Sie hat den Rückgabetyp void, was bedeutet, dass sie nichts zurückgibt, und einen Parameter vom Typ String[] (String-Array), dem man üblicherweise den Namen args gibt.

So definiert man eine Methode:

```
  Rückgabetyp methodenName(Typ1 parameter1, Typ2 parameter2, ...)
  {
    ... Anweisungen ...
  }
```

Bei Java stehen alle Methoden in Klassen. (Fast) alle Anweisungen stehen innerhalb von Methoden.

Es folgt ein Beispiel mit einer void-Methode, die nichts zurück gibt. Angaben wie public oder static stehen immer vor dem Rückgabetyp, aber ihre Reihenfolge ist egal. Man schreibt üblicherweise public vor static.

```
class MethodenBeispiel
{
  public static void main(String[] args)
  {
    System.out.println("Anfang");
    printHallo();
    System.out.println("Ende");
  }

  static void printHallo()
  {
    System.out.println("hallo");
  }
}
```

Das Programm gibt nacheinander Folgendes aus:

```
Anfang
hallo
Ende
```

Ein Programm beginnt mit der ersten Zeile der main-Methode und endet mit der letzten Zeile der main-Methode (*). Es wird also nicht etwa nach System.out.println("Ende") mit der nächsten weiter unten kommenden Anweisung, hier System.out.println("hallo"), weiter gemacht.

Nach System.out.println("Anfang") wird die Methode printHallo() aufgerufen. Der Programmfluss geht also in der Methode printHallo weiter, alle Anweisungen darin, also hier nur die eine, die "hallo" ausgibt, werden abgearbeitet. Dann kehrt der Programmfluss dorthin zurück, wo die Methode aufgerufen wurde und es geht mit der nächsten Anweisung dort weiter.

((*) Ausnahmen, die hier aber nicht relevant sind: statische Initialisierer werden vor der ersten Zeile der main()-Methode ausgeführt. Mit System.exit(0); kann man ein Programm vorzeitig abbrechen. Und wenn mehrere Threads laufen, müssen erst alle beendet werden, bevor das Programm beendet ist - das ist zum Beispiel bei GUIs der Fall.)

Die Reihenfolge der Methoden ist in Java irrelevant. Folgendes Programm ist vollkommen identisch mit dem obigen:

```
class MethodenBeispiel
{
  static void printHallo()
  {
    System.out.println("hallo");
  }

  public static void main(String[] args)
  {
    System.out.println("Anfang");
    printHallo();
    System.out.println("Ende");
  }
}
```

Die Definition einer Methode ist vom Aufruf einer Methode offenbar dadurch zu unterscheiden, dass beim Aufruf der Rückgabetyp nicht davor steht und danach keine geschweifte Klammer und der Methodenrumpf mit den Anweisungen kommt. (Das kann am Anfang verwirrend sein, Definition und Aufruf zu unterscheiden, wenn kein `define` wie in Scheme oder `procedure` oder `function` wie in Pascal vor dem Funktionsnamen steht.)

Nun zu einer Methode, die einen Parameter vom Typ int erwartet, ihn quadriert und das Ergebnis zurückgibt. Ihr Rückgabetyp ist also offensichtlich auch int (das Quadrat einer ganzen Zahl ist eine ganze Zahl).

```
class Quadrat
{
  public static void main(String[] args)
  {
    int zahl = 5; // Kann man natürlich auch von der Tastatur einlesen
    int quadrat = quadriere(zahl);
    System.out.println("Das Quadrat von " + zahl + " ist " + quadrat);
  }

  static int quadriere(int z)
  {
    int q = z*z;
    return q;
  }
}
```

In der Zeile int quadrat = quadriere(zahl) wird also die Methode quadriere aufgerufen. Ihr wird als *Parameter* (oder *Argument*) zahl, also 5, übergeben. In der Methode ist dann also z gleich 5. Das

Quadrat wird berechnet. Mit return wird das Ergebnis zurückgegeben. Das Ergebnis, 25, wir dann also (wieder zurück in der main-Methode) in der Variablen quadrat gespeichert. Und in der nächsten Zeile ausgegeben.

In einer void Methode kann man übrigens als letzte Zeile return (ohne etwas dahinter) schreiben, braucht man aber nicht. return heißt sowohl zurückgeben also auch zurückkehren.

Von einer static-Methode aus kann man nur wieder eine static-Methode (direkt) aufrufen. Um eine nicht-statische Methode aufzurufen, müsste man ein Objekt der jeweiligen Klasse erzeugen, das haben wir hier ja nicht gemacht. Mehr dazu in OO3.

Man hätte die Variablen nicht unterschiedlich nennen müssen, es hätte auch so funktioniert:

```
class Quadrat
{
  public static void main(String[] args)
  {
    int zahl = 5; // Könnt ihr natürlich auch von der Tastatur einlesen
    int quadrat = quadriere(zahl);
    System.out.println("Das Quadrat von " + zahl + " ist " + quadrat);
  }

  static int quadriere(int zahl)
  {
    int quadrat = zahl*zahl;
    return quadrat;
  }
}
```

Die Variablen zahl und quadrat in der main-Methode sind völlig unabhängig von den Variablen in der quadriere-Methode. Sie sind jeweils lokal. Deshalb ist es kein Problem, wenn es in einer Methode Variablen mit demselben Namen wie in einer anderen Methode gibt.

Lokale Variablen heißen auch automatische Variablen, weil sie automatisch auf dem Stack angelegt werden und auch automatisch wieder freigegeben werden. Der Stack ("Kellerspeicher") ist ein bestimmer Teil im RAM (Arbeitsspeicher), eben der, wo lokale Variablen und Parameter (und außerdem Rücksprungadressen) abgelegt werden.

Hier ein Beispiel mit einer nicht-lokalen Variable:

```
class Global
{
  static String s;

  public static void main(String[] args)
  {
    s = "Hallo";
    ausgabe();
    System.out.println(s);
  }

  static String ausgabe()
  {
    System.out.print(s);
    s = " Welt";
  }
}
```

Dieses Programm gibt *Hallo Welt* aus:

1. In main() wird s auf Hallo gesetzt.
2. Dann springt der Programmfluss zur Methode ausgabe().
3. s, welches den Wert Hallo hat, wird ausgegeben.
4. Dann wird s auf Welt gesetzt.
5. Ende der Methode, zurück zu main, wo sie aufgerufen wurde.
6. s nochmal ausgeben, jetzt also Welt.

Da s außerhalb der Methoden definiert ist, kann jede Methode darauf zugreifen.

static Methoden können nur auf static Variablen zugreifen (es sei denn, sie würden erst ein Objekt der Klasse erzeugen, was wir ja nicht tun).

Die Beispielmethoden waren natürlich nicht sehr sinnvoll. Man würde normalerweise nicht eine Zeile ausgeben oder eine so simple Berechnung wie das Quadrieren in eine Methode verlegen. (statische) Methoden haben in erster Linie zwei Zwecke:

1. wiederkehrende Aufgaben zu modularisieren
2. zu strukturieren.

(Nicht-statische Methoden können denselben Zweck haben oder zur Kapselung dienen, siehe OO3.)

Quadrieren ist so trivial, dass man es nicht in einer Methode tun würde. Aber z.B. die Wurzel zu berechnen benötigt sicher viele Zeilen Code. Die kopiert man natürlich nicht an jede Stelle, wo eine Wurzel benötigt wird, sondern schreibt sie in eine Methode, die man dann aufruft.

Aber auch, wenn etwas nur einmal gemacht wird, steckt man es manchmal in eine Methode, damit es übersichtlicher wird. Methoden sind in imparativen Programmiersprachen wie Java normalerweise deutlich länger als in funktionalen Programmiersprachen wie Scheme, wo so manche Funktion nur eine Zeile hat. Aber sie sollten auch nicht zu lang sein. Manche Leute sagen, maximal eine Bildschirmlänge, andere sagen drei Bildschirmlängen, andere meinen, es kommt mehr darauf an, nicht zu tief (mit ifs und Schleifen) zu verschachteln und diese ggf. auszulagern. Eine feste Regel gibt es nicht, es sollte einfach übersichtlich und verständlich bleiben.

Es ist nicht zulässig, Methoden innerhalb anderer Methoden zu definieren.

Parameterübergabe

Primitive Parameter werden beim Übergeben kopiert. Eine Veränderung des Parameters in der Methode verändert nicht die ursprüngliche Variable:

```
class Parameter
{
  public static void main(String[] args)
  {
    int i = 3;
    mache5(i);
    System.out.println(i); // immer noch 3
  }

  static void mache5(int i)
  {
    i = 5;
  }
}
```

Das nennt man *call by value*. Das Gegenteil ist *call by reference*. Nach der genaueren Behandlung von Objekten werden wir darauf zurückkommen, was passiert, wenn ein Objekt als Parameter übergeben wird.

Klassen sind Sammlungen von Methoden

In OO2 haben wir Klassen als komplexe Datentypen kennen gelernt. Aber Klassen können auch Sammlungen von Methoden sein.

```
class MatheHelfer
{
  static int quadrat(int i)
  {
    return i*i;
  }

  static double zins(double kapital, double zinssatz)
  {
    return kapital * zinssatz / 100;
  }

  static int rest(int dividend, int divisor)
  {
    return dividend % divisor;
  }
}
```

In einer anderen Klasse kann man dann auf diese Methoden zugreifen. Bei statischen Methoden einfach mit KlassenName.methodenName.

```
class MatheNutzer
{
  public static void main(String[] args)
  {
    BufferedReader br = new BufferedReader(new StreamInputReader(System.in));
    System.out.print("Anfangskapital: ");
    int kapital = Integer.parseInt(br.readLine());
    System.out.print("Zinssatz: ");
    int zinssatz = Integer.parseInt(br.readLine());
    double ertrag = MatheHelfer.zins(kapital, zinssatz);
    System.out.println("Nach einem Jahr erhalten Sie Zinsen in Höhe von "+ertra
  }
}
```

Mit der JRE kommt eine große Sammlung von Klassen, die API. In der Klasse java.lang.Math sind viele mathematische Methoden definiert. Z.B. Math.sqrt zum Berechnen der Wurzel und Math.pow(basis, exponent) zum Berechnen von "hoch" (nicht etwa ^ benutzen, das ist XOR!). Alle Methoden in Math sind static. Man kann auf sie immer mit Math.methodenName zugreifen. Weil Math im Package java.lang liegt, braucht man Math nicht zu importieren.

```
    double wurzel2 = Math.sqrt(2);
    double einhundert = Math.pow(10, 2);
    double dritteWurzelAusAcht = Math.pow(8, 1.0/3);
```

(Vorsicht Falle: Wenn man im letzten Beispiel 1/3 geschrieben hätte, hätte man 8 hoch 0 ausgerechnet, da 1/3 als Integerdivision 0 ergibt.)

Auch Methoden wie Integer.parseInt, Double.parseDouble sind statische Methoden, aber die Klassen java.lang.Integer und java.lang.Double (sowie Short usw.) enthalten auch viele nicht-statische Methoden.

Konvention: Methodennamen immer klein schreiben. Innere Worte groß (CamelCase). Keine Unterstriche. Bei sonst komplett groß geschriebenen Abkürzungen üblicherweise nur den ersten Buchstaben groß schreiben (bzw. am Wortanfang gar keinen), z.B. convertToHtml, pdfToPs).

Übung 1

Schreibe eine Klasse mit einer main-Methode und mindestens zwei anderen statischen Methoden. Die eine soll void sein und die andere etwas zurückgeben.

Übung 2

Ruf die Java-API auf, geh auf `java.lang` und dann auf `Math`. Benutze einige der Methoden von Math in einem eigenen Programm.

Übung 3

Schreibe eine Klasse Mathe mit einigen statischen Methoden, die etwas wie Quadrat, Wurzel oder Zins berechnen. (Es ist okay, wenn sie nur eine Methode von java.lang.Math aufrufen.) Schreibe eine Klasse MatheNutzer mit einer main-Methode, die die Methoden aus Mathe verwendet.

In der nächsten Lektion werden Arrays eingeführt.