Methoden

Bis jetzt haben wir unseren Programmcode stets in die main()-Methode geschrieben. Das wird auf Dauer viel zu unübersichtlich. Außerdem verstoßen wir so gegen zwei wichtige Prinzipien der Programmierung:

* dem *Single Responsibility Principle (SRP)* und
* *Don't repeat yourself (DRY)*.

Die ursprüngliche Formulierung des SRP stammt von [**Robert C. Martin**](http://blog.cleancoder.com/), der es als ein Prinzip der Objektorientierung einführte und es ein wenig anders meinte, als wir es hier verwenden. Dazu kommen wir, wenn wir uns mit Objektorientierung beschäftigen. Wir können uns aber als wesentliche Prinzipien schonmal merken, dass

* eine Variable genau eine Bedeutung haben soll und niemals für verschiedene Bedeutungen benutzt werden sollte (zwei Bedeutungen == zwei Variablen) und
* eine *Methode* genau eine Sache erledigen sollte.

Zunächst schauen wir uns an, was eine *Methode* überhaupt ist und wie wir sie definieren und verwenden. Angenommen, wir haben ein Programm in der folgenden Form:

public class Methods

{

public static void main(String[] args)

{

int summand1 = 3;

int summand2 = 4;

int summe = summand1 + summand2;

System.out.println(summand1 + " + " + summand2 + " = " + summe); // 3 + 4 = 7

summand1 = 5;

summand2 = 9;

summe = summand1 + summand2;

System.out.println(summand1 + " + " + summand2 + " = " + summe); // 5 + 9 = 14

summand1 = -115;

summand2 = 999;

summe = summand1 + summand2;

System.out.println(summand1 + " + " + summand2 + " = " + summe); // -115 + 999 = 884

}

}

In dieser main()-Methode machen wir drei Mal das Gleiche, wir addieren 2 Summanden und geben das Ergebnis der Berechnung aus. Wir sehen insbesondere doppelten (sogar dreifachen) Code, d.h. wir wiederholen uns. Außerdem geben die vergebenen Namen nur an, wofür die Variablen da sind, aber es gibt keine namentliche Beschreibung von dem, *WAS* wir tun.

**Methodendefinition**

|  |  |
| --- | --- |
| 1  2  3  4  5 | public static void add(int summand1, int summand2)  {  int summe = summand1 + summand2;  System.out.println(summand1 + " + " + summand2 + " = " + summe);  } |

Das wollen wir ändern und laden den sich wiederholenden Code in eine *Methode* aus. Diese Methode nennen wir add():

Betrachten wir diese *Definition einer Methode* genauer:

* In Zeile 1 sehen wir den *Methodenkopf*:
  + Das Schlüsselwort public besagt, dass diese Methode von allen anderen Klassen (die wir noch nicht haben) aufgerufen werden kann. Es handelt sich um eine *öffentliche* Methode. Wir gehen darauf genauer ein, wenn wir uns mit *Sichtbarkeitsmodifizierern* beschäftigen.
  + Das Schlüsselwort static besagt, dass wir diese Methode verwenden (aufrufen) können, ohne eine Objekt der Klasse Methods erzeugen zu müssen. Wir können derzeit eh noch keine Objekte erzeugen, also definieren wir zunächst alle unsere Methoden als static (*statisch*, *Klassenmethode*).
  + Das Schlüsselwort void steht dafür, dass der Aufruf unserer Methode keinen Wert hat, d.h. der Aufruf dieser Methode ist eine Anweisung ohne Nebeneffekt. Wenn die Methode einen Wert haben soll, dann wird hier ein Datentyp eingetragen (sehen wir im nächsten Beispiel).
  + add ist der Methodenname. Hier gelten die Bedingungen, die wir an [**Bezeichner**](http://freiheit.f4.htw-berlin.de/prog1/methoden/variablen/#bezeichner) in Java haben. Methodennamen beginnen stets mit einem Kleinbuchstaben.
  + Nach dem Methodennamen kommen runde Klammern und darin sogenannte *Parameter*. Parameter sind Variablen. Parameter werden in der Methodendefinition deklariert, aber nicht initialisiert. Parameter werden beim Aufruf der Methode initialisiert.
* In den Zeilen 2-5 steht der *Methodenkörper*:
  + Der Methodenkörper ist ein Anweisungsblock. Er beginnt mit einer öffnenden geschweiften Klammer { (Zeile 2) und endet mit einer schließenden geschweiften Klammer } (Zeile 5). Innerhalb dieses Anweisungsblocks können beliebig viele Anweisungen stehen.
  + In Zeile 3 wird unter Verwendung der Werte der Variablen (Parameter) summand1 und summand2 eine Summe gebildet und in der Variablen summe vom Typ int gespeichert.
  + Die Werte der Parameter und der Summe werden in Zeile 4 geeignet auf die Konsole ausgegeben.

Die Definition einer Methode erfolgt immer

* innerhalb einer Klasse und
* außerhalb jeder anderen Methode.

**Methodenaufruf**

In der main()-Methode wird unsere Methode nun aufgerufen. Wichtig ist es zu beachten, dass

* wir exakt den gleichen Namen für die Methode verwenden, wie in der Methodendefinition angegeben (Groß- und Kleinschreibung beachten!) und
* dass der Methode in den runden Klammern Werte für die Parameter übergeben werden. Dabei müssen
  + die Anzahl der Parameter und
  + der jeweilige Typ der Parameter mit dem Aufruf übereinstimmen.

Hier nochmal die gesamte Klasse Methods mit den Aufrufen der add()-Methode in main():

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public class Methods  {  public static void add(int summand1, int summand2)  {  int summe = summand1 + summand2;  System.out.println(summand1 + " + " + summand2 + " = " + summe);  }  public static void main(String[] args)  {  add(3,4);  add(5,9);  add(-115,999);  }  } |

In der main()-Methode wird nun drei Mal unsere neue add()-Methode aufgerufen. Bei jedem Aufruf werden Werte für die Parameter übergeben. Der Aufruf der Methode entspricht einer Anweisung (Semikolon am Ende). Der Aufruf der add()-Methode entspricht keinem Ausdruck, da der Aufruf dieser Methode ohne Wert ist. Dies liegt daran, dass in der Methodendefinition angegeben wurde, dass der Wert der Methode void ist - also kein Wert, kein Typ.

Beachten Sie, dass in der Klasse Methods nun zwei Methoden definiert sind, main() und add(). Die main()-Methode ist die *Programmmethode*, die automatisch ausgeführt wird, sobald wir das Programm starten. Damit die add()-Methode ausgeführt wird, muss sie aufgerufen werden.

**Beachte**

Es werden nur alle Anweisungen ausgeführt, die in der main()-Methode enthalten sind! Wird add() nie in main() aufgerufen, wird add() auch niemals ausgeführt. Die Definition der Methode allein sorgt noch nicht für dessen Ausführung!

**Ausführung des Programms im Detail**

Wir schauen uns die Ausführung des obigen Programms nochmal im Detail an, um die Aufrufe genauer zu analysieren:

1. durch das Starten des Programms wird die main()-Methode aufgerufen (Zeile 9)
2. die erste Anweisung in der main()-Methode ist add(3,4); (Zeile 11)
3. dadurch wird die add()-Methode aufgerufen (Zeile 3)
4. durch den Aufruf werden die Parameter der Methode deklariert und initialisiert, d.h. int summand1 = 3 und int summand2 = 4 (Zeile 3)
5. die erste Anweisung in der Methode add() ist int summe = summand1 + summand2;. dadurch wird die Variable summe deklariert und bekommt den Wert des Ausdrucks summand1 + summand2 initial zugewiesen. Dieser Wert ist 7. (Zeile 5)
6. Es wird die Methode System.out.println() aufgerufen. Der auszugebene String ergibt sich aus summand1 + " + " + summand2 + " = " + summe. Der Wert (vom Typ String) dieses Ausdrucks ergibt sich aus:
   * summand1 + " + " ist ein Konkatenation; das Ergebnis ist "3 + ".
   * "3 + " + summand2 ist ebenfalls eine Konkatenation; das Ergebnis ist "3 + 4".
   * "3 + 4" + " = " ist ebenfalls eine Konkatenation; das Ergebnis ist "3 + 4 = ".
   * "3 + 4 = " + summe ist ebenfalls eine Konkatenation; das Ergebnis ist "3 + 4 = 7".
7. Nach Ausgabe des Strings in Zeile 6 ist die Abarbeitung der add()-Methode beendet. Diese Methode wird verlassen und es wird zurück zur main()-Methode gegangen.
8. die nächste Anweisung in der main()-Methode ist add(5,9); (Zeile 12)
9. dadurch wird erneut die add()-Methode aufgerufen (Zeile 3)
10. durch den Aufruf werden die Parameter der Methode deklariert und initialisiert, d.h. int summand1 = 5 und int summand2 = 9 (Zeile 3)
11. die erste Anweisung in der Methode add() ist int summe = summand1 + summand2;. dadurch wird die Variable summe deklariert und bekommt den Wert des Ausdrucks summand1 + summand2 initial zugewiesen. Dieser Wert ist 14. (Zeile 5)
12. Es wird die Methode System.out.println() aufgerufen. Der auszugebene String ergibt sich aus summand1 + " + " + summand2 + " = " + summe. Der Wert (vom Typ String) dieses Ausdrucks ist "5 + 9 = 14".
13. Nach Ausgabe des Strings in Zeile 6 ist die Abarbeitung der add()-Methode beendet. Diese Methode wird verlassen und es wird zurück zur main()-Methode gegangen.
14. die nächste Anweisung in der main()-Methode ist add(-115,999); (Zeile 13)
15. dadurch wird erneut die add()-Methode aufgerufen (Zeile 3)
16. durch den Aufruf werden die Parameter der Methode deklariert und initialisiert, d.h. int summand1 = -115 und int summand2 = 999 (Zeile 3)
17. die erste Anweisung in der Methode add() ist int summe = summand1 + summand2;. dadurch wird die Variable summe deklariert und bekommt den Wert des Ausdrucks summand1 + summand2 initial zugewiesen. Dieser Wert ist 884. (Zeile 5)
18. Es wird die Methode System.out.println() aufgerufen. Der auszugebene String ergibt sich aus summand1 + " + " + summand2 + " = " + summe. Der Wert (vom Typ String) dieses Ausdrucks ist "-115 + 999 = 884".
19. Nach Ausgabe des Strings in Zeile 6 ist die Abarbeitung der add()-Methode beendet. Diese Methode wird verlassen und es wird zurück zur main()-Methode gegangen.
20. in der main()-Methode gibt es keine weitere Anweisung mehr. Das Programm ist beendet.

**Methode gibt einen Wert zurück**

Unsere Methode add() hat keinen Wert zurückgegeben. Das wurde im Methodenkopf festgelegt, wo wir mit void definiert haben, dass der Aufruf der Methode keinem Wert entspricht. Dies ist typisch für Methoden, die etwas auf die Konsole ausgeben. Alle Methoden, deren Aufgabe es ist, etwas auszugeben, sind (sollten sein) vom Rückgabetyp[1](http://freiheit.f4.htw-berlin.de/prog1/methoden/#fn:1) void.

Jetzt erstellen wir eine Methode computeSum(), die das gleiche macht wie add(), aber mit dem Unterschied, dass diese Methode nichts auf die Konsole ausgibt, sondern die Summe der beiden Parameter an den Aufrufer der Methode *zurückgibt* .

Die Definition dieser Methode sieht dann so aus:

|  |  |
| --- | --- |
| 1  2  3  4  5 | public static int computeSum(int summand1, int summand2)  {  int summe = summand1 + summand2;  return summe;  } |

Zwei ganz wesentliche Unterschiede zur Definition von add() fallen auf:

* Diese Methode hat einen Rückgabetyp (int). Dort, wo bei add() noch void stand, steht bei computeSum() im Methodenkopf int. Damit wird festgelegt, dass der Aufruf der Methode einem Wert entspricht, welcher vom Typ int ist. Der Aufruf dieser Methode ist somit ein Ausdruck!
* Die letzte Anweisung der Methode computeSum() ist eine Anweisung, die mit dem Schlüsselwort return beginnt. Jede Methode, die einen Rückgabetyp hat (also genau nicht void), muss ein solches return enthalten. Dieses return muss die letzte Anweisung in der Methode sein und es muss einen Wert zurückgeben, der von dem Typ ist, der für die Methode als Rückgabetyp definiert wurde. Hier ist es der Wert von summe. summe ist vom Typ int und somit ist return summe; korrekt, da die Methode ja ein int zurückgeben soll.

Aufruf einer Methode, die einen Wert zurückgibt

Unsere Methode computeSum() könnte nun in der main()-Methode wie folgt aufgerufen werden:

computeSum(3,4); // korrekt, aber sinnlos

Ein solcher Aufruf macht aber gar keinen Sinn, weil die Methode selbst ja z.B. nichts ausgibt und somit hat diese Methode gar keinen Effekt. Sinnvoll eingesetzt werden kann eine solche Methode nur als Ausdruck, z.B.:

int sum = computeSum(3,4); // sum wird mit dem Wert 7 initialisiert

System.out.println(computeSum(5,9)); // es wird 14 ausgegeben

Der Aufruf der Methode ist somit ein arithmetischer Ausdruck und kann auch als solcher behandelt werden, z.B. mit anderen arithmetischen Ausdrücken mittels arithmetischer Operatoren zu einem weiteren arithmetischen Ausdruck verknüpft werden.

Hier noch weitere Beispiele für Methoden mit Rückgabe (hier Rückgabe vom Typ boolean):

public static boolean areEqual(int nr1, int nr2)

{

return (nr1 == nr2);

}

public static boolean isDivider(int nr1, int nr2)

{

return (nr1%nr2 == 0);

}

Sie können auch Methoden in Methoden aufrufen. Nehmen wir die beiden Methoden areEqual(int, int) und isDivider(int, int) und angenommen, wir wollen für 2 int-Zahlen prüfen, ob die eine Teiler der anderen ist, aber beide sollen nicht gleich sein, dann können wir folgende Methode schreiben:

public static boolean isDividerButNotEqual(int nr1, int nr2)

{

return (isDivider(nr1, nr2) && !areEqual(nr1, nr2));

}

Das schauen wir uns einmal genauer an:

* wir definieren wieder eine Methode wie gehabt:
  + wir vergeben einen Namen (isDividerButNotEqual) und
  + wir legen fest, dass bei Aufruf der Methode zwei int-Werte übergeben werden müssen ((int nr1, int nr2)).
  + als Rückgabetyp definieren wir boolean, denn wir wollen ja prüfen, ob sich die beiden ganzzahlig teilen, aber nicht gleich sind
* innerhalb der Methode rufen wir die Methode isDivider(nr1, nr2) auf und übergeben dabei unsere Werte für nr1 und nr2. Der Aufruf dieser Methode entspricht einem booleschen Ausdruck, da diese Methode einbooleanzurückgibt (true, wennnr2Teiler vonnr1ist undfalse` sonst - also, wenn nicht)
* außerdem rufen wir die Methode areEqual(nr1, nr2) auf und übergeben dabei ebenfalls unsere Werte für nr1 und nr2. Der Aufruf dieser Methode entspricht ebenfalls einem booleschen Ausdruck, da diese Methode einbooleanzurückgibt (true, wennnr1undnr2gleich sind undfalse` sonst - also, wenn nicht)
* wir wollen aber prüfen, ob sie **nicht** gleich sind, also schreiben wir !areEqual(nr1, nr2) - also die Negation dieses Ausdrucks
* wir wollen prüfen,
  + ob isDivider(nr1, nr2) **UND NICHT** areEqual(nr1, nr2),
  + also isDivider(nr1, nr2) **UND** !areEqual(nr1, nr2),
  + also isDivider(nr1, nr2) && !areEqual(nr1, nr2)
* diesen Wert geben wir zurück

1. Übung Methoden mit Rückgabe1. Übung Methoden mit Rückgabe

**Success**

Wir können uns nun Methoden selber definieren. Die Definition von Methoden erfolgt innerhalb der Klasse, aber außerhalb jeder anderen Methode. Eine Methode kann entweder keinen Wert zurückgeben. Dann ist der "Rückgabetyp"[1](http://freiheit.f4.htw-berlin.de/prog1/methoden/#fn:1) void. Eine solche void-Methode gibt typischerweise etwas auf die Konsole aus. Oder die Methode gibt einen Wert zurück. Dann wird der Datentyp dieses Wertes im Methodenkopf der Methodendefinition angegeben. Die Rückgabe des Wertes erfolgt durch return. Die return-Anweisung muss die letzte Anweisung in der Methode sein. Der Aufruf einer solchen Methode entspricht dann einem Ausdruck. Einer Methode können beliebig viele Parameter übergeben werden. Diese lokalen Variablen werden im Methodenkopf in den runden Klammern durch Komma getrennt deklariert. Bei Aufruf der Methode müssen diesen Variablen Werte übergeben werden (Anzahl und Datentypen müssen bei Methodenaufruf passen).

1. void ist kein Datentyp! Man sagt aber, dass Methoden, die keinen Wert zurückliefern, vom Rückgabetyp void sind. Ganz korrekt ist das also nicht, aber es fehlt ein besserer Ausdruck dafür. [↩](http://freiheit.f4.htw-berlin.de/prog1/methoden/#fnref:1)[↩](http://freiheit.f4.htw-berlin.de/prog1/methoden/#fnref2:1)