Java-Crashkurs

4. Methoden | Benno Hölz, Matthias Ruf | 08.09.2023

sfz®
schülerforschungszentrum
südwürttemberg

Wozu Methoden?

Methoden

- Anreihungen von Befehlen
- können vom Programm aufgerufen werden
  - an verschiedenen Stellen
  - mit verschiedenen Werten (sog. Parameter)
- können Werte zurückgeben

Wozu Methoden?

Methoden

- Anreihungen von Befehlen
- können vom Programm aufgerufen werden
  - an verschiedenen Stellen
  - mit verschiedenen Werten (sog. Parameter)
- können Werte zurückgeben

Wozu Methoden?

- Wiederholter Code kann vermieden werden.
  - dadurch weniger Tipparbeit

  $\Rightarrow$ weniger Fehleranfälligkeit
- Übersichtlichkeit

1. Aufbau einer Methode (void)

Aufbau einer Methode (void)

```
1 void methodenName(int parameter){ // Signatur
2         // Rumpf
3 }
```

Aufbau einer Methode (void)

```java
void methodenName(int parameter){ // Signatur
        // Rumpf
}
```

Methodenname

- gibt der Methode einen Namen
- wird in Java üblicherweise im lowerCamelCase geschrieben:
  - ichBinEineMethode(int methodenParameter){}
- steht in jedem Methodenaufruf

Aufbau einer Methode (void)

```java
void methodenName(int parameter){ // Signatur
        // Rumpf
}
```

Methodenname

- gibt der Methode einen Namen
- wird in Java üblicherweise im lowerCamelCase geschrieben:
  - ichBinEineMethode(int methodenParameter){}
- steht in jedem Methodenaufruf

Parameterliste

- Variablen, die benötigt werden um die Methode aufzurufen
- Definiert **Datentyp**, **Bezeichner** (innerhalb des Methodenrumpfs) und **Anzahl** der Variablen der Parameterliste.

2. Übergabeparameter

Werte übergeben

Übergabeparameter

Über die Parameterliste können einer Methode Variablen übergeben werden.

Beispiel

```
1 void hello(String name){
2         System.out.println("Hallo "+name+"!")
3 }
4
5 public static void main(String[] args){
6         hello("Peter"); // Ausgabe: Hallo Peter!
7 }
```

Call by Value/Reference

Call By Value

Übergabe einer Kopie des Variablenwertes.

```
1 public static void square(int i){
2         System.out.println(i*i);
3 }
4
5 public static void main(String[] args){
6         int i = 5;
7         square(i);
8 }
```

Beispiele

- alle primitiven Datentypen
- Strings

Call by Value/Reference

Call By Reference

Übergabe der Referenz eines Variablenwertes

```
1 public static void add(int[] a){
2         a[1]++;
3 }
4
5 public static void main(String[] args){
6         int[] a = {1,2,3,4};
7         add(a); // a = {1,3,3,4}
8 }
```

Beispiele

- Objekte
- Arrays ($\Rightarrow$ Objekte)

3. Rückgabedatentypen

Rückgabedatentypen

Beispiel

- Integeraddition

```
1 public static int add(int a, int b){
2         return a+b;
3 }
```

- Alternative Methode, welche direkt das Ergebnis ausgibt:

```
1 public static void printAdd(int a, int b){
2         System.out.println(a + b);
3 }
```

Was fällt auf? Was für Unterschiede enthält die Methodensignatur?

Rückgabedatentypen

Beispiel

- Integeraddition

```
1 public static int add(int a, int b){
2         return a+b;
3 }
```

- Alternative Methode, welche direkt das Ergebnis ausgibt:

```
1 public static void printAdd(int a, int b){
2         System.out.println(a + b);
3 }
```

Was fällt auf? Was für Unterschiede enthält die Methodensignatur?

Rückgabe-Datentyp

Eine Methode kann einen Wert **zurückgeben** (jeder Datentyp ist möglich). Es kann aber auch **nichts** zurückgegeben werden (Schlüsselwort `void`).

Rückgabedatentypen

return-Statement

- Jede Methode, die **nicht** `void` als Rückgabewert hat, **benötigt ein** `return`-Statement.

Rückgabedatentypen

return-Statement

- Jede Methode, die **nicht** `void` als Rückgabewert hat, **benötigt ein** `return`-Statement.
- Jede Methode **kann ein oder mehrere** `return`-Statements besitzen

Rückgabedatentypen

return-Statement

- Jede Methode, die **nicht** `void` als Rückgabewert hat, **benötigt ein** `return`-Statement.
- Jede Methode **kann ein oder mehrere** `return`-Statements besitzen
- ein `return`-Statement sieht wiefolgt aus:

```
1 return out;
```

  wobei out ...
  - ... eine Variable sein kann
  - ... ein Ausdruck sein kann
  - ... den passenden Datentyp haben muss.

Rückgabedatentypen

return-Statement

- Jede Methode, die **nicht** `void` als Rückgabewert hat, **benötigt ein** `return`-Statement.
- Jede Methode **kann ein oder mehrere** `return`-Statements besitzen
- ein `return`-Statement sieht wiefolgt aus:

```
1 return out;
```

  wobei out ...
  - ... eine Variable sein kann
  - ... ein Ausdruck sein kann
  - ... den passenden Datentyp haben muss.
- Nach einem `return`-Statement wird der Wert zurückgegeben und die Ausführung der Methode beendet.

  → `void`-Methoden können mit `return` abgebrochen werden

Rückgabedatentypen

Beispiel

```
1  public class Numbers{
2          // Additions-Methode:
3          public static int add(int a, int b){
4                  return a+b;
5          }
6
7          // Aufruf:
8          public static void main(String[] args){
9                  int summe = add(5,3);
10                 System.out.println(summe);
11         }
12 }
```

4. Methoden und Objektorientierung

Methoden in Klassen

Klassen-Methoden

Neben Attributen kann eine Klasse auch Methoden beinhalten.

Beispiel

```
1 public class Auto{
2         private int tank = 50;
3
4         public void fahren(){
5                 this.tank--;
6         }
7 }
```

Methoden in Klassen

Sichtbarkeits-Modifikatoren

- 4 mögliche Modifikatoren:
  - **public**
  - *package*
  - protected
  - private
- Regeln die Sichtbarkeit für andere Klassen

Statische vs. nicht-statische Methoden

Statische Methoden

Statische Methoden gehören **zur Klasse**. Sie können also auch aufgerufen werden, ohne dass ein Objekt der Klasse erzeugt wird.

→ Schlüsselwort `static`

Nicht-statische Methoden

Nicht-statische Methoden hingegen benötigen ein **Objekt**, auf dem sie aufgerufen werden.

Statische vs. nicht-statische Methoden

Beispiel für eine statische Methode

```
1  public class MathHelper{
2          public static int add(int a, int b) {
3            return a + b;
4          }
5
6          public static void main(String[] args) {
7            int result = MathHelper.add(5, 3);
8            System.out.println(result);
9          }
10 }
```

Statische vs. nicht-statische Methoden

Beispiel für eine nicht-statische Methode

```
1  public class Person{
2          private String name;
3
4          // Konstruktor
5          public Person(String name) {
6            this.name = name;
7          }
8
9          // Methode zur Begruessung
10         public void greet() {
11           System.out.println("Ich heisse " + name);
12         }
13
14         public static void main(String[] args) {
15           Person alice = new Person("Alice");
16           alice.greet(); // Ausgabe: Ich heisse Alice
17         }
18 }
```

Kapselung

Kapselung

Kapselung beschreibt das Schützen von Attributen einer Klasse vor (ungewollten) externen Zugriffen.

Kapselung

Kapselung

Kapselung beschreibt das Schützen von Attributen einer Klasse vor (ungewollten) externen Zugriffen.

Kapselung erreicht man wie folgt:

- Attribute auf `private` setzen
- **Getter**- und **Setter**-Methoden verwenden

Kapselung

Beispiel

```
1  public class Person {
2          private String name;
3
4          public Person(String name) { // Konstruktor
5                  this.name = name;
6          }
7
8          public String getName() {
9                  return name;
10         }
11
12         public void setName(String name) {
13                 this.name = name;
14         }
15 }
```

5. Weitere Eigenschaften

Methoden überladen

Überladen

Konstellation zweier oder mehrerer Methoden, welche den gleichen Namen, aber unterschiedliche Parameterlisten haben.

Methoden überladen

Überladen

Konstellation zweier oder mehrerer Methoden, welche den gleichen Namen, aber unterschiedliche Parameterlisten haben.

Möglichkeiten beim Überladen

- Parameterliste unterscheidet sich nur dann, wenn sich die Datentypen der Parameter unterscheiden
  - Nur andere Bezeichner für die Parameter reichen nicht
- Rückgabedatentyp darf sich unterscheiden

Methoden überladen

Beispiele

```
1  public class Add{
2          // Methodenaufrufe in der main-Methode
3          public static void main(String[] args){
4                  int methode1 = add(1,2)   // 1+2   = 3
5                  int methode2 = add(1,2,3) // 1+2+3 = 6
6          }
7
8          // Addiert 2 Ganzzahlen (methode1)
9          public static int add(int a, int b){
10                 return a + b;
11         }
12
13         // Addiert 3 Ganzzahlen (methode2)
14         public static int add(int a, int b, int c){
15                 return a + b + c;
16         }
17 }
```

Ausblick: Rekursionsprinzip

Rekursion

Das direkte oder indirekte Aufrufen einer Methode **von sich selbst**.

Ausblick: Rekursionsprinzip

Rekursion

Das direkte oder indirekte Aufrufen einer Methode **von sich selbst**.

Beipiel Fakultät

- Definition Fakultät:

$$n! = \begin{cases} n \cdot (n-1)! & n > 1 \\ 1 & n \leq 1 \end{cases}$$

- Code der rekursiven Fakultätsmethode:

```
1 public static int fakultaet(int n){
2         if(n <= 1){
3                 return 1;
4         }
5         return n * fakultaet(n-1);
6 }
```

Ausblick: Rekursionsprinzip

Rekursion

Das direkte oder indirekte Aufrufen einer Methode **von sich selbst**.

Beipiel Fakultät

- Definition Fakultät:

$$n! = \begin{cases} n \cdot (n-1)! & n > 1 \\ 1 & n \leq 1 \end{cases}$$

- Code der rekursiven Fakultätsmethode:

```
1 public static int fakultaet(int n){
2         if(n <= 1){
3                 return 1;
4         }
5         return n * fakultaet(n-1);
6 }
```

Wir wollen uns das ersparen

6. Zusammenfassung

Zusammenfassung

- Methoden sind **Anreihungen von Befehlen**
  - können **aufgerufen** werden
  - Können **Übergabeparameter** haben
    → Call by **Reference** vs. Call by **Value**
  - können einen oder keinen **Rückgabewert** haben (Datentyp/`void`)
- Es gibt **statische** und **nicht-statische Methoden**
- Verschiedene **Sichtbarkeits**-Modifikatoren
- Über **Kapselung** können Klassen-Attribute geschützt werden
- Methoden können **überladen** werden
- Methoden können sich selbst aufrufen (**Rekursion**)