Informatik: Methoden

Ziele

- Den Aufbau von Methoden (Kopf, Rumpf, Parameter, Rückgabewert) verstehen und anwenden können
- Den Zusammenhang von lokalen Methodenvariablen und Parametern verstehen können
- Das Überladen von Methoden anwenden können
- Das Kommentieren von Methoden richtig einsetzen können
- Methoden in eigenen Klassen sammeln können

Beispiel

```
public class Testprogramm;
{
   public static void main(String[] args) {
      int a = 7; int b = 10;
      for (int i = a; i < b; i = i + 1)
          tuewas(i, "Hallo");
      boolean ergebnis = tuewas(a + 1);
   }
   public static void tuewas() {
      System.out.println("Nix is fix");
   }
   public static void tuewas(int i, String s) {
      int a = 3 + i;
      System.out.println(s + a);
   }
   public static boolean tuewas(int i) {
      boolean ret = false;
      if (i > 7) {
          tuewas();
          ret = true;
      }
      return ret;
   }
}
```

Methoden

- definieren Abläufe
- müssen zu Klassen gehören
- bestehen aus *Kopf* und *Rumpf*

```
public static void tuewas() {                    Methodenkopf
   System.out.println("Nix is fix");
                                                 Methodenrumpf
}
```

- können *Parameterliste* (input) haben

```
public static void tuewas() { … }
public static void tuewas(int i, String s) { … }
public static void main(String[] args) { … }
```

- können eigene, *lokale Variablen* besitzen, können aber nicht auf lokale Variablen *anderer Methoden* zugreifen

```
public static void tuewas(int i, String s) {
   int a = 3 + i;
   a = a + b;
}
```

FEHLER weil b nicht in derselben Methode deklariert oder in Parameterliste enthalten

- liefern *keinen Rückgabewert*, falls **void**
  können aber auch einen Rückgabewert (output) liefern

```
public static boolean tuewas(int i, String s) {
   boolean ret = false;
   …
   return ret;
}
```

- können *aufgerufen* werden. Dabei ist die Reihenfolge der übergebenen Parameter und ihre Typen wichtig. Die Namen der Parameter sind irrelevant

```
AUFRUF:              tuewas(  3, "hallo");

public static void tuewas(int i, String s) {
   …
}
AUFRUF: tuewas("hallo", 3);
AUFRUF: tuewas(3.0, "hallo");
AUFRUF: tuewas("hallo");
int a = 3;
AUFRUF: tuewas(a,"hallo");
AUFRUF: tuewas(a + 3, "hallo");
AUFRUF: System.out.println(tuewas(3,"hallo"));
```

FEHLER weil Methode keinen Rückgabewert liefert, der weiter verarbeitet werden könnte

- Änderungen der Parameter haben *keine Auswirkung* in der aufrufenden Methode

```
public static void main(String[] args) {
   int i = 3;
   erhoeheUmEins(i);
   System.out.println(i);
}
public static void erhoeheUmEins(int i) {
   i = i + 1;
}
```

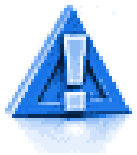
- können andere Methoden *derselben Klasse* aufrufen

```
public static boolean tuewas(int i) {
   …
   tuewas();
   …
}
```

- In einer Klasse können mehrere Methoden *mit demselben Namen* definiert werden (**Überladung** engl. **overloading**), müssen sich aber in der Parameterliste unterscheiden.
  Compiler sucht die richtige Methode lt. Parameterliste aus

```
public static void tuewas() { … }
public static void tuewas(int i, String s) { … }
public static boolean tuewas(int i) { … }
```

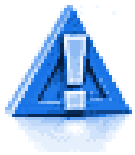
- können öffentlich (**public**) oder privat (**private**) sein



Programmierregeln

- Methoden müssen eingerückt werden
- Methodennamen beginnen mit Kleinbuchstaben
- In der ersten Zeile eines Methodenrumpfes mit Rückgabe muss deklariert werden `int` (?) `ret = 0;` der Name der Rückgabevariable muss `ret` sein
- Im Methodenrumpf einer Methode mit Rückgabe darf nur in der letzten Zeile ein einziges `return ret;` stehen

Kommentierregeln



Vor Methodenkopf müssen in *Dokumentationskommentaren* (/**) folgende Punkte beschrieben werden

- *Funktionsweise* der Methode (auch anhand von Beispielen)
- *Sonderfälle* auf die Methode reagieren muss
- *Parameter* und ihre Bedeutung
- Welche *Rückgabewert* werden geliefert

```
/**
 * Liefert einen String zurück der aus den übergebenen Zeichen zeichen besteht.
 * Wie viele Zeichen der String enthalten soll, wird in anzahl übergeben. So
 * liefert z. B. der Aufruf generiereString('*',3) den String "***"
 * zurück.
 * anzahl muss > 0 sein, ansonsten wird null zurück geliefert
 * @param zeichen das Zeichen welches im zurückgegebenen String wiederholt
 * auftreten soll
 * @param anzahl der Zeichen, welche im String eingefügt werden sollen
 * @return den String, der so viele Zeichen enthält, wie in anzahl übergeben
 * wurde
 */
public static String generiereString(char zeichen, int anzahl) {
   String ret = null;
   if (anzahl > 0) {
      ret = "";
      for (int i = 0; i < anzahl; i = i + 1)
          ret = ret + zeichen;
   }
   return ret;
}
```

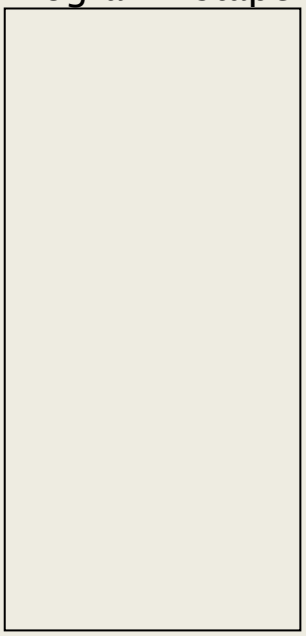
Beispiel: Welche Ausgabe liefert folgendes Programm?

```
public class WelcheAusgabe
{
   public static void main(String[] args) {
      int x = 3;
      System.out.println(eins(eins(eins(x + 1))));
      zwei(x, x + 2);
   }

   public static int eins(int x) {
      int ret = x + 2;
      System.out.println(ret);
      return ret;
   }

   public static void zwei(int a, int b) {
      int x = a; int y = b;
      System.out.println(eins(a + b) + eins(b));
   }
}
```

Programmstapel



Ergebnis: 6 8 10 10 10 7 17

Beispiel: Rekursiver Aufruf

```
public class BerechnungFakultaet
{
   public static void main(String[] args) {
      int z = 10;
      System.out.println("Die Fakultät von " + z + " ist " +
          fakultaet(z));
   }

   public static int fakultaet(int z) {
      int ret = 1;
      if (z > 1)
          ret = z * fakultaet(z - 1);
      return ret;
   }
}
```

Methoden in eigenen Klassen sammeln

```
public class MeinStringManipulator
{
   public static String sperren(String s) {
      String ret = "";
      if (s.length() > 0)
          ret = s.charAt(0) + " " + sperren(s.substring(1));
      return ret;
   }
   …
}
```

```
public class MeinHauptprogramm
{
   public static void main(String[] args) {
      String s = "String sperren";
      System.out.println(MeinStringManipulator.sperren(s));
   }
}
```

- Methoden welche ähnliche Aufgaben erledigen in eigener Klasse sammeln
- Klasse welche Methoden sammelt, muss keine `main`-Methode enthalten
- Bei Methodenaufruf muss Klassenname angeführt werden `MeinStringManipulator.sperren(s)`
- Klasse welche Methoden enthält muss im *selben Ordner* wie Hauptprogramm abgespeichert sein