Methoden

Javakurs 2014, 3. Vorlesung

Sebastian Schuck

basierend auf der Vorlage von
Magdalena Rätz

`wiki.freitagsrunde.org`

4. März 2014

Inhaltsverzeichnis

Inhaltsverzeichnis

Was haben wir gestern kennengelernt?

- grundlegender Programmaufbau
- Datentypen:
  int, **boolean**, String
- Variablen deklarieren und Wertzuweisungen:
  int a = 42;
- Bedingte Anweisungen:
  if (condition){ } **else** { }
- Arrays:
  char[] wort = {'w','o','r','t'}
- Schleifen:
  for (**int** i = 0, i < wort.length, i+=2){ }
  while (condition){}

Inhaltsverzeichnis

Motivation für Methoden

- helfen, den Code zu strukturieren
- erlauben Wiederverwendung von Code
  - Redundanz wird vermieden
  - Fehleranfälligkeit sinkt
  - *siehe Beispiel*

Motivation für Methoden

- helfen, den Code zu strukturieren
- erlauben Wiederverwendung von Code
  - Redundanz wird vermieden
  - Fehleranfälligkeit sinkt
  - *siehe Beispiel*

Methoden

- können direkt einen Effekt haben, aber kein Ergebnis liefern.
  System.out.println("Hallo"); → Text auf dem Bildschirm
- können keinen direkten Effekt haben, aber ein Ergebnis liefern
  int ersteSumme = berechneSumme(erstesArray);
  → Ergebnis von `berechneSumme` wird in `ersteSumme` gespeichert.

Aufbau einer Methode

```
1 public static <Rueckgabetyp> <Name>(<Parameterliste>) {
2     <Anweisungen>
3 }
```

- Methodenkopf in Zeile 1
  - **public**: Sichtbarkeit der Methode
  - **static**: Unterscheidung Objektmethode vs. statische Methode
  - **Rückgabetyp**: Alle Datentypen möglich
    → spezieller Rückgabetyp **void** bedeutet "Keine Rückgabe"
  - **Name**: Name der Methode (z.B. main)
  - **Parameterliste**: beliebig viele Parameter
    → jeweils Parametertyp und -name mit Komma getrennt
    → Parameterliste kann auch leer sein
- Methodenrumpf in Zeile 2
  → beliebig viele Anweisungen

Flaecheninhalt Variante 1

```
1 public static int berechneFlaeche(int a, int b) {
2     int flaeche = a*b;
3     return flaeche;
4 }
```

Flaecheninhalt Variante 1

```
1 public static int berechneFlaeche(int a, int b) {
2     int flaeche = a*b;
3     return flaeche;
4 }
```

Flaecheninhalt Variante 2

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

Flaecheninhalt Variante 1

```
1 public static int berechneFlaeche(int a, int b) {
2     int flaeche = a*b;
3     return flaeche;
4 }
```

Flaecheninhalt Variante 2

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

sayHello

```
1 public static void sayHello(String name) {
2     System.out.println("Hallo "+name);
3     // bei void ist kein return notwendig
4  }
```

1. Welche Variablen kennt meine Methode?
   - "Klassenvariablen": in der Klasse definierte Variablen
   - die übergebenen Parameter
2. Was kann man machen, wenn man zusätzliche Variablen braucht?

1. Welche Variablen kennt meine Methode?
   - "Klassenvariablen": in der Klasse definierte Variablen
   - die übergebenen Parameter
2. Was kann man machen, wenn man zusätzliche Variablen braucht?
   $\rightarrow$ dann definiert man "lokale Variablen" in der Methode

1. Welche Variablen kennt meine Methode?
   - "Klassenvariablen": in der Klasse definierte Variablen
   - die übergebenen Parameter
2. Was kann man machen, wenn man zusätzliche Variablen braucht?
   → dann definiert man "lokale Variablen" in der Methode

Merke!

- Variablen sind in genau dem Block gültig, in dem sie deklariert worden sind.
- "Klassenvariablen" wurden innerhalb der äußersten beiden geschweiften Klammern deklariert.
  → daher sind sie im gesamten Programm gültig
  - zumindest gilt das bis hier hin – später können Programme aus mehreren Klassen bestehen.

Verschattung

lokale Variablen können (globale) Klassenvariablen *verschatten*

- lokale Variablen sind Parameter oder innerhalb der Methode deklarierte Variablen
- lokale Variablen können genauso heißen wie Klassenvariablen
- im Falle der Verschattung:
  - hat man keinen Zugriff auf den Wert der Klassenvariable
  - kann keine (versehentliche) Änderung des Werts der Klassenvariable vorgenommen werden
    $\rightarrow$ ohne Verschattung kann man auf Klassenvariablen zugreifen

Beispiel

```
1  class Geometrie {
2      // Methode berechneFlaeche
3      public static void main(String[] args) {
4        int a = 3;
5        int b = 2;
6        int umfang = 2*a+2*b;
7        int c = berechneFlaeche(a,b);
8        System.out.println("Rechteck " + a + ", " + b
9                           + ": Flaeche = " + c + ", Umfang = " + umfang);
10     }
11 }
```

Was wird ausgegeben, wenn berechneFlaeche so definiert wird?

Fall 1 (Standardfall)

```
1 static int berechneFlaeche(int a, int b) {
2       return a*b;
3 }
```

Beispiel

```
1  class Geometrie {
2      // Methode berechneFlaeche
3      public static void main(String[] args) {
4          int a = 3;
5          int b = 2;
6          int umfang = 2*a+2*b;
7          int c = berechneFlaeche(a,b);
8          System.out.println("Rechteck " + a + ", " + b
9                              + ": Flaeche = " + c + ", Umfang = " + umfang);
10     }
11 }
```

Was wird ausgegeben, wenn berechneFlaeche so definiert wird?

Fall 2

```
1  static int berechneFlaeche(int a, int b) {
2      int umfang = a+1;
3      return a*b;
4  }
```

Beispiel

```
1  class Geometrie {
2      // Methode berechneFlaeche
3      public static void main(String[] args) {
4          int a = 3;
5          int b = 2;
6          int umfang = 2*a+2*b;
7          int c = berechneFlaeche(a,b);
8          System.out.println("Rechteck " + a + ", " + b
9                              + ": Flaeche = " + c + ", Umfang = " + umfang);
10     }
11 }
```

Was wird ausgegeben, wenn berechneFlaeche so definiert wird?

Fall 3

```
1  static int berechneFlaeche(int a, int b) {
2      a = a+1;
3      return a*b;
4  }
```

Beispiel

```
1  class Geometrie {
2      // Methode berechneFlaeche
3      public static void main(String[] args) {
4        int a = 3;
5        int b = 2;
6        int umfang = 2*a+2*b;
7        int c = berechneFlaeche(a,b);
8        System.out.println("Rechteck " + a + ", " + b
9                           + ": Flaeche = " + c + ", Umfang = " + umfang);
10     }
11 }
```

Was wird ausgegeben, wenn berechneFlaeche so definiert wird?

Fall 4

```
1  static int berechneFlaeche(int b) {
2      a = a+1;
3      return a*b;
4  }
```

Overloading - Überladen von Methoden

Es kann mehrere (unterschiedliche) Methoden mit dem selben Namen geben $\rightarrow$ *Overloading*.

Einschränkung:

Methoden mit dem selben Namen müssen trotzdem eindeutig identifizierbar sein.

- mittels unterschiedlicher Parameteranzahl
- mittels unterschiedlichen Parametertypen

Unterschiedliche Rückgabetypen reichen für eine Eindeutigkeit nicht aus!

Overloading - Beispiel

Die Klasse Geometrie soll für Kreise erweitert werden.

Overloading - Beispiel

Die Klasse Geometrie soll für Kreise erweitert werden.

berechneFlaeche für Kreise?

```
1 static double berechneFlaeche(int radius, int pi) {
2     return pi*radius*radius;
3 }
```

Overloading - Beispiel

Die Klasse Geometrie soll für Kreise erweitert werden.

berechneFlaeche für Kreise?

```
1 static double berechneFlaeche(int radius, int pi) {
2     return pi*radius*radius;
3 }
```

- Der Standardfall ist bereits eine Methode `berechneFlaeche` mit zwei Parametern vom Typ **int**.
- Die Variablennamen der Parameterliste können nicht zur Unterscheidung von Methoden verwendet werden.
- $\rightarrow$ daher keine eindeutige Indentifizierung möglich

Overloading - Beispiel

Die Klasse Geometrie soll für Kreise erweitert werden.

berechneFlaeche für Kreise?

```
1 static double berechneFlaeche(int radius, int pi) {
2     return pi*radius*radius;
3 }
```

- Der Standardfall ist bereits eine Methode `berechneFlaeche` mit zwei Parametern vom Typ **int**.
- Die Variablennamen der Parameterliste können nicht zur Unterscheidung von Methoden verwendet werden.
- $\rightarrow$ daher keine eindeutige Indentifizierung möglich

besser!

```
1 static double berechneFlaeche(int radius) {
2     return java.lang.Math.PI*radius*radius;
3 }
```

Fragen für die Vorgehensweise

- Welche Parameter werden benötigt?
- Welches Ergebnis soll geliefert werden?
- Wie komme ich mit den Parametern auf das Ergebnis?

häufige Fehlerquellen

- Vergessen eines Statements im Methodenkopf
- falsche Typen in Parameterliste
- falscher Rückgabetyp
- **return**-Statement vergessen

- Kann man Methoden innerhalb von anderen Methoden deklarieren?
  $\rightarrow$ **Nein**, aber man kann andere Methoden aufrufen/verwenden.

Inhaltsverzeichnis

Aufruf von Methoden innerhalb der Klasse

Für die Belegung der Parameterliste kommt es zur Übergabe von:

- vorhandenen Variablen
- festen Werten

ohne Rückgabewert

```
1 public static void main(String[] args) {
2     int a = 3;
3     meineMethode(a);
4 }
```

mit Rückgabewert

```
1 public static void main(String[] args) {
2     int b = meineMethode(3);
3 }
```

Aufruf von Methoden aus anderen Klassen

ohne Rückgabewert

```
1 public static void main(String[] args) {
2     int meineVar = 3;
3     MeineKlasse.meineMethode(meineVar);
4 }
```

mit Rückgabewert

```
1 public static void main(String[] args) {
2     int meineVarA = 3;
3     int meineVarB = MeineKlasse.meineMethode(meineVarA);
4 }
```

Besonderheit: rekursive Methoden

Rekursion nennt man die Definition einer Funktion durch sich selbst

- z.B. bei der rekursiven Definition von Folgen
- Rekursive Methoden rufen sich immer wieder selbst auf, bis die Abbruchbedingung erreicht ist.

$$\begin{aligned}(a_n) &= 2 \cdot a_{n-1} \\ (a_0) &= 1\end{aligned}$$

Diese Technik wird auch in der Informatik verwendet. Dabei ist die Abbruchbedingung bzw. die Abbruchbedinungen besonders wichtig.

Beispiel: Berechnung der Fakultät

- Bildungsvorschrift: $n! = n \cdot (n-1)!$
- Abbruchbedingung: $0! = 1$

Fakultaet rekursiv

```
1  public class Fakultaet{
2      public static void main(String[] args){
3       System.out.println(fakultaet(4));
4      }
5
6      // nur positive Werte erlaubt
7      public static int fakultaet(int n) {
8          if (n == 0) {
9              return 1;
10         }
11         return n*fakultaet(n-1);
12     }
13 }
```