II.2.2 Methoden, Unterprogramme und Parameter

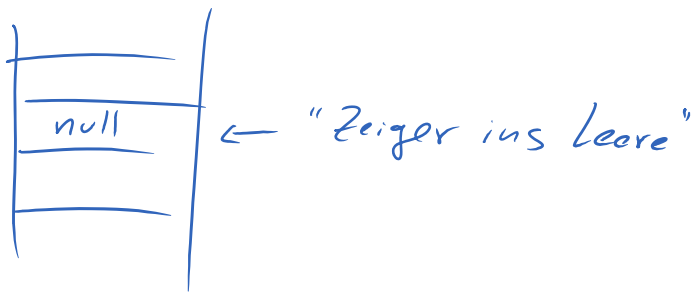
Mittwoch, 9. November 2016 09:30

Methode: Unterprogramm mit Parametern

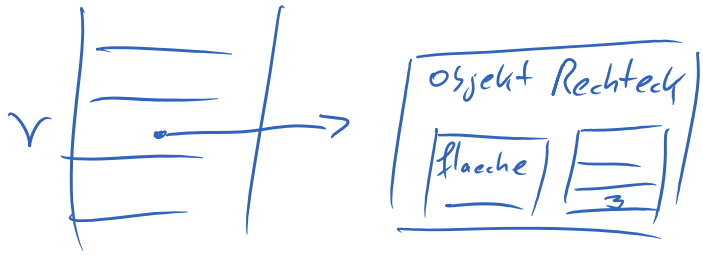
- Über Namen der Methode kann dieses Unterprogramm überall aufgerufen werden.
- Beim Aufruf der Methode werden ihre formalen Parameter durch die aktuellen Parameter "ersetzt".
- Nach Ende der Methode wird das Programm an der Aufrufstelle fortgesetzt.
- Der Typ der aktuellen Parameter muss dem Typ der formalen Parameter entsprechen (Typen müssen gleich sein oder es muss implizite Typkonversion vom Typ der aktuellen Parameter zum Typ d. formalen Parameter gehen).

```
public class Rechteck {
  double laenge, breite;
  int strichstaerke = 3;
  double flaeche() { ... }
}
```

Rechteck r;



r = new Rechteck();



"static": klassische, nicht objektorientierte Methode, die nur von der Klasse abhängt, nicht vom Objekt

Ablauf beim Methodenaufruf: Call-by-value Parameterübergabe

- Ausdruck im aktuellen Parameter wird ausgewertet (Bsp: ergist 1570.22)
- Formale Parameter wird auf den Wert des aktuellen Parameters gesetzt (Bsp: Kapital = 1570.22)
- Methodenrumpf wird ausgeführt, bis man Ende erreicht (bei void-Methoden) bzw. bis man return-Anweisung erreicht (bei anderen Methoden)
- Ausdruck der return-Anweisung wird ausgewertet und an die aufrufende Stelle zurückgeliefert.

Methoden mit Rückgabetyp ≠ void: Funktionen

" " = void: Prozeduren

Sinn von Prozeduren:

- Ausgabe (z.B. drucke)
- Seiteneffekte

Call-by-value Parameterübergabe

- aktueller Parameter wird ausgewertet (s == 2.1)
- Wert d. aktuellen Parameters wird in formalen Parameter d. Methode kopiert. (r = s, d.h. r == 2.1)
- Änderungen des formalen Parameters der Methode bewirken Keine Änderung des aktuellen Parameters (nach Ende der Methode f ist immer noch s == 2.1)

Diese Art der Parameterübergabe wird in Java für primitive Datentypen verwendet.

Speicherverwaltung beim Methodenaufruf

- Block: { Anw.1; ... ; Anw.n } "Stack frame"
  Blöcke können geschachtelt sein.
- Wenn Java einen Block betritt, wird ein Speicherbereich (never) auf dem Laufzeitkeller (runtime stack) angelegt, in dem die Werte der Variablen abgelegt werden, die in diesem Block deklariert werden.
- Beim Eintritt in neuen Block, wird der zugehörige Frame oben auf dem Laufzeitkeller angelegt (in der Zeichnung wachsen die Keller nach unten, d.h. "oben" im Sinn der

Wachstumsrichtung)

- Man sucht Variablen erst im inneren Block, danach im äußeren Block etc.
- Speicherverwaltung bei Methodenaufruf ist ähnlich wie bei Blöcken (denn der Rumpf d. Methode ist auch ein Block).
- Bei Methodenaufruf könnten Variable in unteren Frames den gleichen Namen wie Var. in oberen Frames haben. Sichtbar sind hier aber nur die Frames der aufgerufenen Methode.

Call-by-reference Parameterübergabe

- aktueller Parameter ist Variable
- Formaler Parameter wird Verweis / Referenz auf den aktuellen Parameter.
- Jede Änderung des formalen Parameters bewirkt dann auch eine Änderung des aktuellen Parameters.

In vielen Programmiersprachen kann der Programmierer zwischen call-by-value u. call-by-reference wählen.

In Java: bei nicht-primitiven Datentypen enthalten aktueller + formaler Parameter Referenz auf das Objekt.

Wenn das Objekt in der aufgerufenen Methode verändert wird, dann ist dies als Seiteneffekt in der aufrufenden Methode sichtbar.

⇒ bei nicht-primitiven Datentypen hat Java eine Art call-by-reference.

Java macht eigentlich immer cbv-Parameterübergabe, aber da nicht-primitive Objekte über Referenzen gespeichert sind, entspricht dies einer eingeschränkten Form von cbr.

Wenn r auf ein neues Rechteck-Objekt gesetzt wird, (↑ formaler Parameter) dann hat dies in Java keine Auswirkung auf den aktuellen Parameter s.

(Komplettes call-by-reference ist in Java nicht möglich.)

Schlechteste Form der Parameterübergabe:

globale Variablen

```
public static int x;

public static int f(...) {
    :
    x = x+1;
    :
}
```

```
}
public static void main(...) {
        :  <- x
    f(...)
        :  <- x
}
```

Sinn von Prozeduren (void-Methoden):

- Ausgabe (z.B. drucke)
- Seiteneffekte (z.B. sortiere)

Vararg-Parameter

andere Schreibweise für Array-Parameter

z.B: addiere(2,3,4) entspricht

addiere(new int[] {2,3,4})

Statische Attribute und Methoden

- Jedes Rechteck-Objekt hat eine eigene laenge, breite, strichstaerke, flaeche.
  Zugriff über das Objekt: r.laenge
  r.flaeche()

~~Rechteck.laenge~~

Diese Eigenschaften sind nicht-statisch.

- Manche Variablen + Methoden sollen zu jeweiligen Objekt

- Manche Variablen + Methoden sollen v. jeweiligen Objekt abhängen, andere sollen nur von der Klasse abhängen.

Bsp: Man will mitprotokollieren, wie oft eine Flächenberechnung stattfindet.
⇒ statische Variable flaechenberechnung

Ist Eigenschaft der Klasse:
Rechteck.flaechenberechnung

Ausgabe des Bsp-Programms:
0
1
2

Bisher: Klassen, in denen alles static war ← Sammlung von Unterprogrammen

oder Klassen, in denen nichts static war ← Daten-strukturen

Jetzt: Klassen können beides enthalten

Aufruf von statischen/nicht-statischen Attribute v. Methoden:

- wenn nichts davor steht (kein ... .):

Attr./Methode des aktuellen Objekts (bei: nicht-statischen Attr./Meth)

nur in nicht-statischen Methoden/Attr. mögl.

Attr / Methode der aktuellen Klasse (bei statischen Attr./Methoden)

- ansonsten:

  Var . steht ein

  Objekt (bei nicht-statischen Meth./Attr.)

  z.B. r.laenge, r.flaeche(), ...

  Klasse (bei statischen Meth./Attr.)

  z.B. Sort.drucke(), Rechteck.flaechenberechnung, ...

Auf statische Eigenschaften darf man auch über Objekte zugreifen:
r.flaechenberechnung

Welche Methoden sollten statisch sein?
Alle Methoden, die Eigenschaften des Objekts ausdrücken, sollten nicht-statisch sein (d.h., alle Methoden, die auf die Objekt-attribute zugreifen).
z.B. flaeche, toString

Aufruf r.toString()

Wozu statische Attribute?

Sind insbesondere sinnvoll, um Konstanten zu vereinbaren.

Sonderfall: Aufzählungstypen

Dies sind Klassen mit wenigen Objekten, bei denen jedes Objekt d. Klasse mit einer eigenen Konstante (d.h. einer eigenen statischen Variable) bezeichnet wird.

- Für enum-Klassen erzeugt Java bestimmte vordef. Methoden, z.B. toString, ordinal (Index-Position im entsprechenden Array), ...
- Aufzählungstypen können natürlich weitere Methoden enthalten.
- In switch-Anweisungen kann man Fallunterscheidungen nicht nur für int, char, String, sondern auch für Aufzählungs-

typen durchführen.

Gültigkeit von Bezeichnern

Welche Bezeichner sind wo sichtbar, falls sie gleich heißen.

Gründe für Verwendung der gleichen Bezeichner an verschiedenen Stellen im Prog:

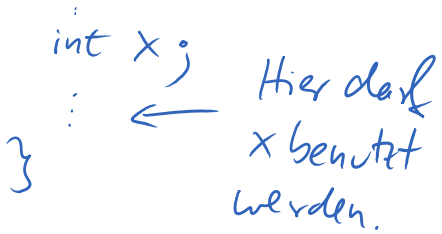
- Modularität (versch. Prog-Teile werden von versch. Programmierern entwickelt).
- Sinnvolle Wiederverwendung von Bezeichnern kann Lesbarkeit erhöhen.

Regeln

- Jeder Bezeichner muss deklariert werden. Deklaration gehört zu dem innersten Block, in dem sie steht.

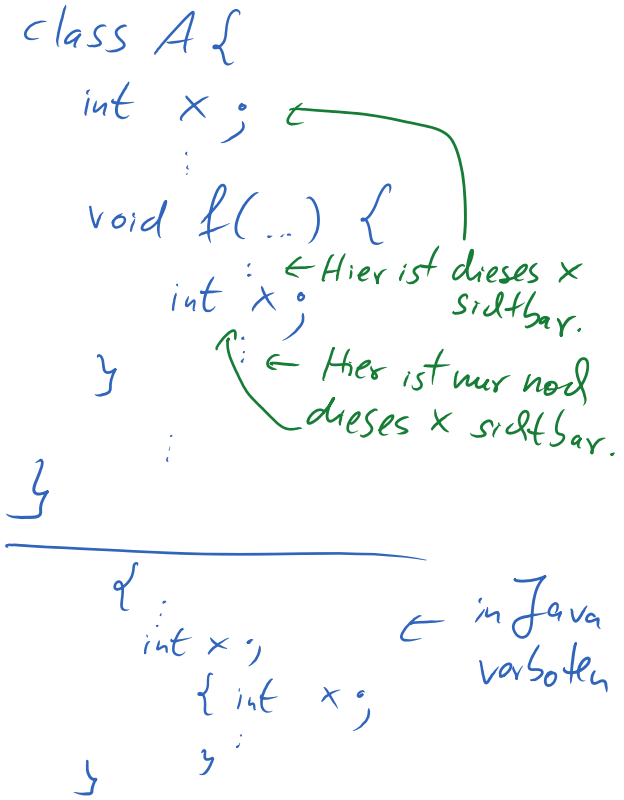
```
{ ⋮
  {
    int x;   ← gehört zu dem inneren Block
  }
}
```

- Bezeichner kann erst nach seiner Deklaration benutzt werden. { ⋮



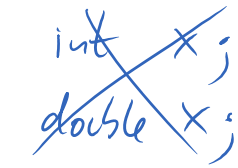
Ausnahme: Klassen und Methoden dürfen schon vor ihrer Deklaration benutzt werden.

- Deklaration gilt bis zum Ende des (inneren) Blocks, der die Dekl. enthält.
- Namensgleiche Bezeichner im inneren Block überdecken Bezeichner im äußeren Block.



- In einem Block müssen alle Bezeichner verschieden sein, es sei denn, sie bezeichnen

Verschiedenartige Programm-elemente (z.B. Klassen, Methoden, Variablen).



verboten

```
int x;
void x() {
    ⋮
}
class x {
    ⋮
}
```

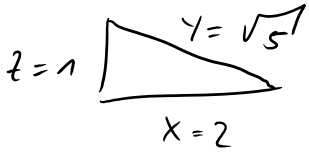
erlaubt

- Mehrere Methoden dürfen den gleichen Namen haben, falls ihre formalen Parameter hinreichend unterschiedlich sind (Überladen von Methoden → später).

Dreiecks-Bsp

- Statische Var. x gilt in gesamter Klasse Gueltigkeit.
- Aber: formaler Parameter x der Methode main überdeckt die statische Var.
- Objektvariablen x, y, z gelten in gesamter Klasse Dreieck. Aber sie werden von den for-

malen Parametern der Methode
"setze" überdeckt.
d.x ist hingegen die Eigenschaft x des Dreiecks d.
⇒ d wird auf dieses Dreieck gesetzt:



- In der Methode flaeche( ) sind x, y, z in der ersten Zuweisung die Objektvar.

Man sollte Mehrfachverwendg v. Bezeichnern nur sinnvoll einsetzen. Sonst werden Prog. unverständlich.